REPORT SAMSO-TR-75-255, VOL. IV

A019 111

GENERALIZED TRAJECTORY SIMULATION

Volume IV: Numerical Operators

TRAJECTORY ANALYSIS PROGRAMMING DEPARTMENT
Information Processing Division
Engineering Science Operations
The Aerospace Corporation
El Segundo, Calif. 90245

15 April 1976

Final Report

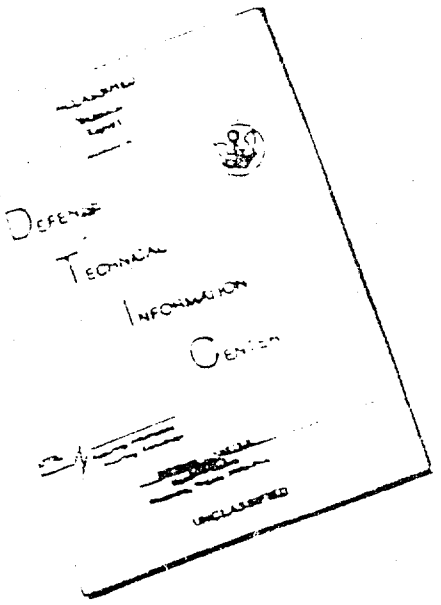
Prepared for

AIR FORCE ROCKET PROPULSION LABORATORY
AIR FORCE SYSTEMS COMMAND
Edwards Air Force Base, Calif. 93523

and THE AEROSPACE CORPORATION

SPACE AND MISSILE SYSTEMS ORGANIZATION
AIR FORCE SYSTEMS COMMAND
Los Angeles Air Force Station
P.O. Box 92960, Worldway Postal Center
Los Angeles, Calif. 90009

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

This final report was submitted by The Aerospace Corporation, El Segundo CA 90245, under Contract F04701-75-C-0076 with the Space and Missile Systems Organization, P.O. Box 92960, Worldway Postal Center, Los Angeles CA 90009. It was reviewed and approved for The Aerospace Corporation by D.A. Schermerhorn and A.R. Sims, Engineering Science Operations. The Air Force Project Engineer was Lt. Joe Hildreth, AFRPL/MKCD.

This technical report has been reviewed and is approved for publication. Publication of this report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

Gerhard E. Aichinger
Technical Advisor
Contracts Management Office

FOR THE COMMANDER

Frank J. Bane
Chief, Contracts Management Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: SAMSO-TR-75-255, Vol IV	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): The Generalized Trajectory Simulation System. Volume IV. Numerical Operators.	5. TYPE OF REPORT & PERIOD COVERED: Final Report, 1-1-75 to 1-1-76
6. PERFORMING ORG. REPORT NUMBER: TR-0076(6666)-1-Vol IV	
7. AUTHOR(s): Trajectory Analysis Programming Department, Information Procession Division, Engineering Science Operation	8. CONTRACT OR GRANT NUMBER(s): F04701-75-C-0076
9. PERFORMING ORGANIZATION NAME AND ADDRESS: The Aerospace Corporation, El Segundo, California 90245	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Space and Missile Systems Organization, Air Force Systems Command, Los Angeles, California 90045	12. REPORT DATE: 15 April 1976 13. NUMBER OF PAGES: 104
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Air Force Rocket Propulsion Laboratory, Air Force Systems Command, Edwards Air Force Base, Calif. 93523	15. SECURITY CLASS. (of this report): Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CDC 6000/7000	Nonlinear Constrained Parameterization Optimization	Optimal Control
Modularized Program	Optimization	Constraint Solving Techniques
Software System	Nonlinear Least Squares	Trajectory Optimization
Numerical Operators	Search Techniques	Performance Analysis
Nonlinear Programming		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Generalized Trajectory Simulation (GTS) system provides a vehicle design and trajectory simulation capability. GTS is written in FORTRAN and is compatible with CDC 6000/7000 series computer systems. User-oriented input data specifications, computational efficiency, diverse program applicability, and convenient program modifications have been primary considerations in the design of the GTS system. The trajectory simulation

DD FORM 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 KEY WORDS (Continued)

Vehicle Sizing
Vehicle Design
Vehicle/Trajectory Optimization
Numerical Integration
Trajectory Simulation
Boost Vehicle Simulation
Reentry Vehicle Simulation
Dynamic System Simulation
Post Flight Reconstruction
Interpolation

20 ABSTRACT (Continued)

capability can accommodate diverse types of vehicle configurations, flight profiles, and mission objectives. Additionally, the GTS system contains an extensive vehicle sizing capability and a state-of-the-art optimization capability.

This volume documents the GTS library of optimization, integration, and interpolation operators. Included with the description of each operator is a description of the input data required to execute the operators and recommendations concerning the types of problems for which each operator is best suited.

PREFACE

This volume, the fourth of five volumes that describe the Generalized Trajectory Simulation (GTS) system, concerns the GTS library of numerical operators, including integration, optimization, and interpolation operators. The remaining volumes are:

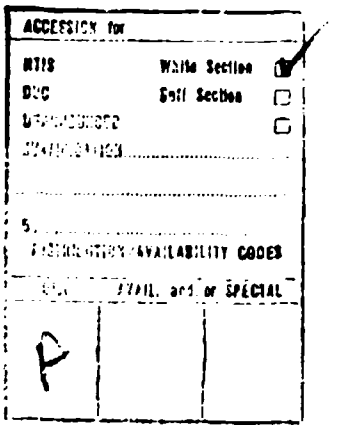
Volume I: GTS Overview. This volume provides the potential user with an overview of GTS, including a summary of the major operational capabilities and structural design of the GTS system.

Volume II: GTS Usage Guide. This volume serves as a general usage guide for GTS and includes a set of example problems, a comprehensive description of the Generalized Trajectory Language, and a discussion of the trajectory simulation control. In addition, a set of appendices contains a master reference list for all volumes and supplementary information to aid the user in defining his problem.

Volume III: GTS Flight Dynamic Models. This volume concerns the GTS library of flight mechanics and flight dynamics models utilized for trajectory simulations.

Volume V: GTS Weight Estimation Models for Sizing Applications. This volume documents the GTS library of weight estimation models utilized for sizing applications.

This report was prepared by J. L. Searcy. The author acknowledges the beneficial contributions and suggestions made by D. S. Meder, J. T. Betts, and G. B. Green.



CONTENTS

		Page
1.	INTRODUCTION	1-1
2.	OPTIMIZATION AND SEARCH CAPABILITY	2.1-1
	2.1 Overview	2.1-1
	2.1.1 Problem Formulation	2.1-1
	2.1.2 Data Specification	2.1-3
	2.1.3 Function Generator Error Response	2.1-4
	2.1.4 Restart Capability	2.1-5
	2.1.5 Optimal Control Problems	2.1-6
	2.1.6 Recommendations for Usage	2.1-7
	2.2 Optimization and Search Operators	2.2-1
	2.2.1 UOPTIM	2.2-1
	2.2.1.1 General Description	2.2-2
	2.2.1.2 Equality Constrained Algorithm	2.2-3
	2.2.1.3 Basis Determination	2.2-6
	2.2.1.4 Example	2.2-6
	2.2.1.5 Special Usage	2.2-7
	2.2.1.6 Interpretive Output from UOPTIM	2.2-9
	2.2.1.7 Diagnostic Output	2.2-10
	2.2.2 UBEST	2.2-12
	2.2.2.1 Preliminary Developments	2.2-13
	2.2.2.2 Unconstrained Optimization Algorithm	2.2-14
	2.2.2.3 Determination of the Basis	2.2-15
	2.2.2.4 Constraint Phase	2.2-16
	2.2.2.5 The Nonlinear Programming Algorithm	2.2-17
	2.2.2.6 Example	2.2-18
	2.2.2.7 Special Usage	2.2-20
	2.2.2.8 Interpretive Output from UBEST	2.2-22
	2.2.2.9 Diagnostic Output from UBEST	2.2-23
	2.2.3 USCHN	2.2-25
	2.2.3.1 Basic Procedure	2.2-26

2.2.3.2 Update Logic 2.2-29
2.2.3.3 USCHN Algorithm 2.2-30
2.2.3.4 Interpretive Output from USCHN 2.2-31
2.3 Optimization Input 2.3-1
2.3.1 Optimization Input Format 2.3-1
2.3.2 Problem Definition Models (PROBDEF) 2.3-2
2.3.2.1 Problem Definition Model 1 (PRBDFM1) . . . 2.3-3
2.3.3 Objective Function Models (OBJFTN) 2.3-8
2.3.3.1 Objective Function Model 1 (OBJFNM1) 2.3-8
2.3.4 Constraint Models (CONSTR) 2.3-11
2.3.4.1 Constraint Model 1 (CNSTRM1) 2.3-11
2.3.5 Independent Variable Models (INDVAR) 2.3-17
2.3.5.1 Independent Variable Model 1 (VARM1) 2.3-17
2.4 Optimization and Search Output 2.4-1
2.4.1 Optimization Input Summary 2.4-1
2.4.2 Iteration Summary 2.4-2

3. INTEGRATION OPERATORS 3-1
3.1 Trajectory Simulation 3.1-1
3.2 Integration Methods 3.2-1
3.2.1 Runge-Kutta, Fixed Step (4-th Order) 3.2-1
3.2.2 Adams-Moulton, Fixed Step (m-th Order) 3.2-2
3.2.3 Adams-Moulton, Variable Step (m-th Order) . . . 3.2-8
3.3 Recommendations for Usage 3.3-1
3.4 Integration Input Models (INTGRA) 3.4-1
3.4.1 Integration Model 1 (INTGRM1) 3.4-1
3.5 Integration Output 3.5-1

4. INTERPOLATION OPERATORS 4-1
4.1 Interpolation Formulas 4.1-1
4.1.1 Univariate Interpolation 4.1-1
4.1.2 Multivariate Interpolation 4.1-3
4.2 Interpolation/Integration Interaction 4.2-1

REFERENCES R-1

FIGURES

		Page
2.2-1	UOPTIM Example	2.2-1
2.2-2	UBEST Example	2.2-19
3.2-1	The 4-th Order Runge-Kutta Integration Method	3.2-3
3.2-2	Adams-Moulton Integration Method	3.2-6
3.2-3	Summary of Derivative Evaluation.	3.2-7

SECTION 1

INTRODUCTION

This volume documents the GTS library of numerical operators. These operators are distinguished from the engineering models documented in Volumes III and V in that they implement a numerical algorithm for the solution of a mathematical problem rather than modeling a physical phenomenon. The three major categories of operators are optimization and search operators, integration operators, and interpolation operators.

To provide the user with an effective and efficient method for solving his individual problem, several operators are available within each category. Furthermore, no a priori restrictions are made concerning the operator that may be selected for a given application. That is, any numerical operator may be specified in conjunction with any particular model configuration that the user has specified.

This independence of the operators and models allows the user considerable flexibility in defining problems for solution. For example, the optimization operators may be specified in conjunction with an integration operator for a trajectory optimization problem, or the same optimization operator may be specified for a vehicle design problem which does not require a trajectory simulation. All the numerical operators are compatible with the GTS input language. Thus, any Generalized Trajectory Language (GTIL) capability such as GTL FORTRAN routines, tabular input, or the equivalence option may be specified at the user's convenience to define an optimization problem.

This volume documents the operators that are available within each category. The discussion of each operator contains a user-oriented description of the algorithm that is embodied in the operator. A more complete mathematical description of each algorithm can be found in the listed references. Clearly, within each category not all operators are equally well suited for all applications. Consequently, included with the description of each operator is an indication of the types of problems for which each operator might be best suited. Also included with the discussion of each operator is a description of the input data required to execute the operator and a description of any output obtained from the operator.

SECTION 2

OPTIMIZATION AND SEARCH CAPABILITY

The GTS optimization and search capability is described in this section. First, an overview of the capability is presented. A description of the individual operators, the input data required to execute these operators, and the output obtained as a part of the optimization process are then presented in succeeding sections. Examples of the application of the optimization and search capability are presented in Volume II.

2.1 Overview

The GTS optimization and search capability is a valuable analytical tool which is an integral part of the GTS system. This capability can be applied to a diverse set of problems. This diversity of application is achieved by permitting the complete library of flight dynamics models and weight estimation models to be available as a part of the definition of an optimization or search problem. Furthermore, the GTS system provides several optimization and search operators, each of which may be applied to several types of problems. Thus, the user is able to specify the problem formulation and operator which best satisfies his requirements for accuracy and efficiency.

The information contained in this section is applicable to all operators and is intended to enable the user to apply effectively the GTS optimization and search capability to his individual application. Included is a complete list of the optimization and search operators which are available in the GTS system and recommendations concerning the class of problems for which each operator is best suited.

2.1.1 Problem Formulation

The GTS optimization capability may be applied to solving general nonlinear constrained parameter optimization problems, search problems, or nonlinear least squares problems. Optimal control problems which can be posed as parameter optimization problems can also be solved. For

purposes of definition, a parameter optimization problem can be described as follows: determine the values of n parameters, or independent variables $(x_1 \ldots x_n)$, such that the scalar function $f(x_1, \ldots, x_n) = f(x)$ is optimized (i.e., maximized or minimized). The function f(x) is referred to, among other names, as the objective function, the performance index, performance criteria or cost function. Additional restrictions may be imposed on the problem so that the variables are to be determined not only to optimize the objective function, but also to satisfy equality constraints of the form

$$c_i(x_1, \ldots, x_n) = 0 \qquad i = 1, \ldots, k$$

and/or inequality constraints of the form

$$c_i(x_1, \ldots, x_n) \geq 0 \qquad i = k+1, \ldots, m$$

Example 3 of Volume II illustrates these concepts. In this example, the objective function is vehicle weight; the independent variables are three burn times, four reorientation angles, and two pitch rates; and the constraints are four equality constraints which define a final orbit. Thus, the problem is to minimize the vehicle weight as a function of the nine specified parameters subject to a set of four equality constraints which define a final orbit.

Search problems are a special case of parameter optimization problems for which an objective function is not explicitly defined. Rather, for search problems the intent is to determine the values of n parameters which satisfy a set of constraints. The number of independent variables may be greater than or equal to the number of constraints. Root solving problems are a subclass of search problems for which the number of variables equals the number of constraints. Example 2 of Volume II is an example of a root solving problem. The constraints are that the longitude and latitude of impact should be equal to specified values. The independent variables or targeting parameters are the launch azimuth and a pitch rate.

An optimization problem whose objective function can be written as the sum of squares of the constraints (residuals) is called a least squares problem. In general, the number of constraints is greater than or equal to the number of variables, and it is not expected that the residuals will be zero at the solution.

A least squares problem of this type can be solved using the optimization operators, by posing a problem with equality constraints and no explicit objective function. Hence, the GTS optimization and search capability may be applied to several types of problems. These options are discussed in more detail in Section 2.3.2.

All the optimization and search operators discussed in Section 2.1 are operational as a part of the GTS system. Thus, the complete GTS model library, the GTL input language, and the GTS process control capability may be employed to define optimization and search problems. This flexibility permits many diverse types of problems to be posed for solution within the GTS system. Potential problems include boost trajectory problems (see example 3 of Volume II), orbital transfer problems (see example 4 of Volume II), or trajectories that include both boost and reentry phases. Vehicle design problems (see example 9 of Volume II) illustrate that an optimization problem need not require a trajectory simulation. Furthermore, as illustrated by example 10 of Volume II, problems may be formulated independent of any flight dynamics models.

2.1.2 Data Specification

The data required to define an optimization or search problem is specified via GTL. A specific format for the definition of optimization and search problems is part of GTL. A complete description of this input format is given in Section 2.2. In addition, all GTL capabilities (see Volume II), such as the tabular input format, event specification, and GTL FORTRAN routines, may be utilized for defining an optimization or search problem. Briefly, the input for optimization and search problems is in the model type--model format. These optimization input model types reflect the major components of an optimization problem (i.e., objective function, constraints and independent variables), and consequently, these model types are independent of a particular operator requested to solve the problem. That is, all the optimization input models described in Section 2.1 are compatible with all of the optimization

operators described in Section 2.2. Furthermore this independence of input models and operators implies that the user may choose the input format which is most convenient for his particular problem, regardless of the operator requested to solve the problem. Also, once a problem has been defined, a different operator may be requested to solve the problem without redefining the input data.

2.1.3 Function Generator Error Response

As a part of an optimization application, the condition may be encountered that the GTS system is unable to compute the values of the constraints or the objective function for a specified set of values of independent variables. For example, consider a trajectory optimization application for which the values of the independent variables are such that an abnormal trajectory termination (e.g., a negative altitude) occurs. From the user's viewpoint, this situation may be tolerable if the optimization process can recover from the current difficulty, continue the optimization process, and ultimately obtain a solution. Alternately, the user may feel that the inability of the program to compute all desired quantities is symptomatic of a fundamental error in the problem formulation. For this case the user would prefer that the optimization process be terminated and that the problem be studied before continuing the optimization process.

The GTS optimization system is cognizant of these potential situations, and the following logic concerning the inability of the function generator to complete a function or gradient evaluation has been implemented into the GTS optimization system. First, there are two situations which result in the program terminating when such a point is encountered. One is that the user requests, via input, that the job be terminated whenever the program is unable to complete an evaluation of the objective function or constraints. The second such situation is that the initial point is not a computable point. That is, the system does not attempt to determine a feasible point if the first point is not feasible; rather, it terminates and requests that the user provide such

information. Here a feasible point is defined to be a point where all desired evaluations can be made.

Otherwise, the program will attempt to continue if a nonfeasible point is encountered. Except for the initial point, a nonfeasible point can occur in two different circumstances, and the program response is different in each case. First, the nonfeasible point is a prediction made by the optimization operator. For this case, a message is printed indicating the situation and a return is made to the optimization operator with the information that an evaluation was not possible at the last point. The optimization operator must then provide a new prediction. Second, the nonfeasible point is a perturbed trajectory requested in the process of providing numerical derivative information. For this case, the current perturbation size is changed and a new set of perturbed evaluations are attempted. A message indicating the situation and the new values of the perturbation size is printed.

Finally a note of caution--this error response capability is available for the user's convenience in situations where a nonfeasible point is a temporary anomaly and the rather simple logic described above is sufficient to overcome the difficulty. The logic is probably inadequate to handle more complicated situations such as a problem in which the solution lies near a region of non-feasibility. Furthermore, this capability should not be a substitute for an adequate problem formulation.

2.1.4 Restart Capability

The algorithms embodied in the UOPTIM and UBEST operators are recursive in nature. That is, successive estimates of the solution are based not only on the information obtained at the most recent evaluation, but on information obtained at all previous evaluations. Consequently, the algorithms are generally more efficient if they are allowed to continue uninterrupted to a solution rather than requiring the program to begin anew at a set of intermediate points. It is not always practical to permit the program to proceed

uninterrupted to a solution, or the user may want to analyze intermediate results. For these situations a restart capability has been incorporated into the program. This restart capability permits recursively generated information to be saved at the completion of a job if a solution has not been obtained. A subsequent run may access the information generated by the original run, thereby eliminating the need to regenerate this information. The input data required to save this information and subsequently access this information is discussed in Section 2.3.2 .

2.1.5 Optimal Control Problems

In addition to the nonlinear programming problems and search problems previously discussed, optimal control problems may be posed for solution on GTS. For purposes of definition, assume a set of state equations of the form

$$\frac{dx}{dt} = f(t, x, u) \qquad x(t_o) = x_o$$

where x and f are n-vectors and u is an m-vector. Also, assume a performance index of the form

$$J(u) = G(t_o, t_f, x(t_o), x(t_f)) + \int_{t_o}^{t_f} g(t, x, u)\, dt$$

The problem is to choose the set of functions u , referred to as the control functions, such that the functional J(u) is minimized. In addition, constraints of the form

$$c_i(t, x, u) = 0 \qquad i = 1, \ldots, j$$

$$c_i(t, x, u) \geq 0 \qquad i = j+1, \ldots, k$$

may be imposed on the system. For optimal control problems, as contrasted with optimization and search problems, the problem solution requires the

determination of a time dependent function u(t) rather than a finite set of parameters. Moreover, the constraints are time dependent functions. In addition, the initial time t_o, or final time t_f may be parameters to be determined. Example 10 of Volume II is an illustration of an optimal control problem. The control functions are the rates of change of mechanical power u_i of the individual power plants. The objective is to minimize the function J, subject to the constraint that the total power produced equal the demanded load.

A general method that may be implemented within the GTS system for the solution of optimal control problems is to approximate the control functions by a finite set of parameters and then to apply the existing nonlinear programming operators to this resulting finite dimensional optimization problem. Again, example 10 of Volume II illustrates this technique. For this example, the set of parameters which approximate the rate of change of mechanical power is the value of these rates at a specified set of time points. This approximation technique is easily implemented via GTL tabular input and the GTL equivalence option. The constraint

$$|u_i(t)| \leq U_i$$

is approximated by a finite set of constraints that require each of the tabular values of the rate of change of mechanical power to satisfy the above constraints. Thus, a time varying constraint has been replaced by a discrete set of constraints.

Certainly, different techniques for approximating the control functions and specifying the constraints may be more desirable for specific problems, and the user has the option of designing a parameterization technique which suits his problem.

2.1.6 Recommendations for Usage

Potential difficulties common to all optimization and search operators are mentioned prior to presenting specific recommendations concerning the individual operators. When defining the input data for a search or

optimization problem, the user must insure that the particular model configuration specified does compute the objective function and the constraints. In addition, the user must insure that the independent variables are valid inputs for the models that are to be executed. For example, for problem number 9 of Volume II, RANGLS is not a valid objective function for the specified model configuration since a trajectory simulation is not being performed.

All the optimization and search operators implicitly assume that the computed quantities (i.e., the objective function and constraints) satisfy continuity and convexity requirements. These requirements are generally very difficult to verify analytically, and it is not expected that the user do so. The user should, however, formulate his problems such that known discontinuities in the function and derivatives are avoided.

For trajectory simulations, the user must accept some inaccuracy associated with the integration process. A given integration accuracy that may be acceptable for the user's analysis, however, may not be acceptable for a simulation which is a part of an optimization or search problem. This situation is especially true for optimization and search problems which obtain derivative information via numerical perturbations. Recommendations for the usage of integration operators are given in Section 4.3.

In addition, for trajectory cases the user should realize that an optimization or search operator may, potentially, request a trajectory simulation with values of the independent variables which results in an event sequence that is different from the one encountered in the nominal case. The user must either formulate the problem in such a manner that this situation presents no difficulty or he must eliminate this problem by the specification of the range of the independent variables or by the specification of the event criteria.

For the convenience of the user, a list of the optimization and search operators available in GTS is given below. Recommendations concerning the types of applications to which each operator might be best applied are

also presented. These recommendations are only to serve as general guides, not absolute rules. Specific knowledge or experience that a user may have concerning an individual problem would supersede these recommendations. The algorithms are undergoing a continuing process of modification and improvement. Furthermore, new algorithms may be introduced into GTS. Consequently, the user should be sure that his selection of an operator is based on the most recent information that is available.

1. USCHN -- USCHN is only applicable to search problems. Furthermore, its effectiveness is generally limited to smaller dimensional problems, roughly less than 10 variables and 10 constraints. USCHN does not require an evaluation of partial derivatives at each point. Consequently, it is more efficient than the other algorithms for those problems where it is applicable.

2. UOPTIM -- UOPTIM is applicable to general nonlinear constrained parameter optimization problems, search problems, least squares problems, and root solving problems. For constrained parameter optimization problems, the UOPTIM algorithm generates a set of intermediate points which satisfy the constraints. Consequently, UOPTIM is preferred for those problems in which it is useful to obtain intermediate points that satisfy the constraints. For all problems the UOPTIM algorithm requires gradient information at every point; consequently, it will be less efficient than USCHN for problems for which USCHN is applicable.

3. UBEST -- UBEST is applicable to general nonlinear constrained parameter optimization problems, search problems, least squares problems, and root solving problems. For constrained parameter optimization problems, UBEST generates a set of intermediate points which are near the optimum of the objective function but which violate the constraints. Consequently, UBEST is preferred for those problems in which it is useful to obtain intermediate points near the optimum of the objective function. For search or least squares problems, the UBEST operator is preferred to UOPTIM, especially for cases which have a nonzero minimum. UBEST requires

gradient information at every point and is less efficient than USCHN for problems for which USCHN is applicable.

2.2 Optimization and Search Operators

All of the optimization and search operators available in the GTS program are discussed in this section. The description of each operator includes a brief, heuristic outline of the optimization or search algorithm and a list of the informative and diagnostic messages that are output by the operator. These messages are intended to permit the user to follow the progress of the algorithm and to analyze the results that are obtained.

2.2.1 UOPTIM

The optimization operator UOPTIM is described in the following sections. This operator is designed to solve the following problem. Determine the n-vector x that optimizes (i.e., maximize or minimize) the scalar function

$$f(x) = f(x_1, x_2, \ldots, x_n) \tag{1}$$

subject to the equality constraints

$$c_i(x) = 0 \quad i = 1, 2, \ldots, k \tag{2}$$

and the inequality constraints

$$c_i(x) \geq 0 \quad i = k+1, k+2, \ldots, m \tag{3}$$

The functions $f(x)$ and $c_i(x)$ are assumed to be twice continuously differentiable in the region

$$x_L \leq x \leq x_U \tag{4}$$

where x_L and x_U are specified upper and lower bounds. These bounds determine a region of computability and unlike constraints cannot be violated during the iterative process.

As special cases of this problem, the UOPTIM operator is designed to solve unconstrained optimization problems in which there are no constraints ($m = 0$), nonlinear root solving problems, search problems, and nonlinear least squares problems.

The following sections give a brief, user-oriented description of the UOPTIM algorithm. A more complete development of the algorithm may be found in Ref. 1.

2.2.1.1 General Description

Assume the optimization problem defined by Eqs. (1), (2), and (3) has a solution x^*. At x^* the following conditions hold:

$$c_i(x^*) = 0 \qquad i = 1, 2, \ldots, k$$

and for

$$i = k+1, k+2, \ldots, m$$

either

$$c_i(x^*) = 0$$

or

$$c_i(x^*) > 0$$

That is, at the solution an inequality constraint is either satisfied as an equality constraint, or it is satisfied as a strict inequality constraint and, as such, imposes no restrictions on the problem. For definiteness, we shall refer to the set of constraints B^* that is satisfied as equality constraints at the solution as the basic set of constraints.

$$B^* = \{c_i \mid c_i(x^*) = 0 \qquad i = 1, 2, \ldots, m\}$$

Clearly, B^* contains all equality constraints and a subset of the inequality constraints. Hence, the optimization problem defined above can be solved if the following two subproblems can be solved. First, determine the basic set of constraints B^*. Second, solve the following equality constraint problem:

optimize $f(x)$

subject to

$$c_i(x) = 0 \quad c_i \in B^*$$

To solve the first problem, a sequence of estimates of the basic set of constraints is made. Each of these estimates of the basic set of constraints is referred to as a basis. Each basis contains all equality constraints and a subset of the inequality constraints. The total number of constraints in each basis is restricted to be less than or equal to n. As each basis, B, is specified, an associated equality constrained optimization problem is posed. After a solution to this equality constrained problem is found, a new basis estimate is made by adding and/or deleting inequality constraints from the existing basis. The constraint addition and deletion logic is formulated such that if at the solution of a posed equality constrained problem no constraints can be added to or deleted from the current basis, then the current basis is the basic set of constraints. Therefore, the current point is the solution to the overall optimization problem.

2.2.1.2 Equality Constrained Algorithm

Assume a basis estimate has been determined. This section describes the associated equality constrained optimization problem that is posed and the method used to solve this problem. The next section describes the constraint addition and deletion logic used to obtain a new basis estimate.

Consider the function

$$\bar{f}(x) = f(x) + rP(x) \tag{5}$$

where $f(x)$ is the objective function, r is a nonnegative scalar referred to as the penalty parameter, and the function $P(x)$ is defined by

$$P(x) = \sum_{c_i \notin B} [c_i^-(x)]^2 \tag{6}$$

where

$$c_i^-(x) = \min[0, c_i(x)]$$

The UOPTIM algorithm poses the following constrained problem for solution:

$$\text{optimize } \bar{f}(x)$$

subject to

$$c_i(x) = 0 \quad c_i \in B$$

This equality constrained problem is solved by the following elimination of variables technique. Assume that c_i, $i = 1,2,\ldots,r \leq n$ are the constraints in the current basis. Then define a new set of variables u_i, $i = 1,2,\ldots,n$, in the following manner. For $i = 1,\ldots,r$ let $u_i = c_i(x)$. For $i = r+1,\ldots,n$, set $u_i = x_i'$, where x_i' is chosen among the original set $x_1 \ldots x_n$. Hence, we have defined a set of functions $\gamma_i(x)$, $i = 1,\ldots,n$, such that $u_i = \gamma_i(x)$, where

$$u_i = \gamma_i(x) = c_i(x) \quad i = 1,2,\ldots,r$$

$$u_i = \gamma_i(x) = x_i' \qquad i = r+1,\ldots,n$$

If we require that the Jacobian matrix $J = \frac{\partial \gamma}{\partial x}$ is nonsingular, then there is an implicitly defined family of functions Γ_i, $i = 1,\ldots n$, such that $x_i = \Gamma_i(u_1, u_2,\ldots,u_n)$, $i = 1,\ldots,n$. Now consider the function g defined by

$$g(u_1,\ldots,u_n) = \bar{f}\,(\Gamma_1(u_1,\ldots,u_n),\ldots,\Gamma_n(u_1,\ldots,u_n))$$

The problem of optimizing $\bar{f}(x)$ subject to $c_i(x) = 0$, $i = 1,\ldots,r$ is equivalent to optimizing $g(u_1, u_2,\ldots,u_n)$ subject to $u_1 = u_2 = \ldots = u_r = 0$. Equivalently, this problem reduces to optimizing

$$h(u_{r+1},\ldots,u_n) = g(0,0,\ldots,0,u_{r+1},\ldots,u_n)$$

Thus, the equality constrained optimization problem with n independent variables has been reduced to optimizing a function h of $n-r$ variables with no constraints.

Note that the $n-r$ optimization variables u_i, $i = r+1, \ldots, n$, are a subset of the original set of independent variables. The remaining r variables from the set $x_1 \ldots x_n$ are determined so that the r constraints

$$u_i = c_i(x) = 0 \quad i = 1, 2, \ldots, r$$

are satisfied. This completely defines the vector x.

Therefore, the equality constrained optimization problem has been "factored" into the two problems of an unconstrained optimization problem of $n-r$ variables and a root solving problem for a system of r variables and r constraints. Currently, the root solving problem is accomplished by minimizing the associated function

$$\phi(x) = \frac{1}{2} \sum_{c_i \in B} [c_i(x)]^2$$

Consequently, the equality constrained problem has been reduced to two lower dimensional unconstrained minimization problems.

The following basic technique is used to solve both of these unconstrained minimization problems. Let f represent the function to be minimized. That is, $f = h$ for the unconstrained minimization problem and $f = \phi$ for the root solving problem. The algorithm then minimizes f by making a series of iterative improvements to an initial estimate x^0, according to the formula

$$x^{i+1} = x^i - \rho^i s^i \quad i = 0, 1, 2, \ldots$$

The search vector s^i is computed by

$$s^i = -K^i \nabla f(x^i) \quad i = 1, 2, \ldots$$

where $\nabla f(x^i)$ is the gradient of f at x^i and K^i is an i-th approximation of the inverse of the Hessian matrix, or matrix of second partial derivatives of f. The scalar ρ^i is determined by a one dimensional search procedure.

At the point x^{i+1}, a new estimate of the inverse of the Hessian matrix K^{i+1} is made based on the values of $f(x^{i+1})$, $f(x^i)$, $\nabla f(x^{i+1})$, $\nabla f(x^i)$ and K^i. Once K^{i+1} is determined, the procedure is to be repeated until the specified convergence criteria are satisfied. This scheme is quasi-second order, in that convergence is quadratic if the objective function is quadratic near the solution.

2.2.1.3 Basis Determination

Assuming that the equality constrained optimization algorithm described in the previous section obtains a solution to the posed problem, we describe the method by which constraints are added to, or deleted from, the current basis to form a new basis. If the current point is a solution to the posed problem, an attempt is made to add all violated constraints to the basis. If this is not possible, then the constraints are ranked in order of their "violation." For the UOPTIM algorithm, the "violation" of a constraint is defined to be a linear estimate of the distance from the current point to the nearest point where the constraint is satisfied. Recalling that a basis may contain at most n constraints, as many constraints as possible are added to the basis in order of their "violation."

The constraint deletion logic attempts to determine the one constraint that, if released from the basis, will result in the greatest improvement in the function f consistent with the requirement that the other constraints in the current basis remain satisfied. The current logic permits only one constraint to be released. Furthermore, the constraint deletion logic is consistent with the Kuhn-Tucker necessary condition for a solution. If at the solution to an equality constrained problem there are no violated constraints and no constraints can be deleted from the current basis, then the current basis is the basic set of constraints and the current point is a solution.

2.2.1.4 Example

As an illustration of the UOPTIM algorithm, consider the following simple problem:

$$\text{minimize} \quad f(x_1, x_2) = x_1^2 + x_2^2$$

$$\text{subject to:} \quad c_1(x_1, x_2) = -(x_1^2)/4 + x_2 + 4 \geq 0$$

$$c_2(x_1, x_2) = 2x_1 + x_2 - 8 \geq 0$$

Figure 2.2-1 illustrates the contours of the objective function, as well as the constraint boundaries. The violated regions for the constraints have been crosshatched. For this problem assume that constraint $c_1(x_1, x_2)$ is in the initial basis. Assume the initial point was specified to be $x = (4, 4)$ (labeled (1) in the figure). Initially, a factorization is obtained which specifies that the variable x_1 is the constraint solving variable and variable number 2 is the optimization variable. Then a solution of the constraints is found as a function of the constraint solving variable. The point (5.6568, 4.0) (labeled (2)) is such a point. Next, the function $f(x)$ is minimized subject to the constraints in the basis, c_1 in this case. The point (3.6673, -0.6377) (labeled (3)) satisfies this condition. Now constraints are added and/or deleted from the current basis. Clearly, $c_2(x)$ must be added to the basis, and the algorithm determines that $c_1(x)$ can be released from the basis. Thus, a new equality constrained problem of optimizing $f(x)$ subject to $c_2(x) = 0$ is posed for solution. The point (3.2, 1.6) (labeled (4)) is then determined to be the solution to this problem. At this point there are no violated constraints, and, furthermore, no constraint can be released from the basis. Thus, the basic set of constraints has been determined, and the point (3.2, 1.6) is the solution to the problem.

2.2.1.5 Special Usage

The following input quantities which are specific to the operator UOPTIM can be specified as a part of the problem definition model PRBDFM1 input (Section 2.3.2.1). However, it is recommended that these quantities not be changed from the preset values without knowledge of the algorithm and a demonstrated requirement for an alternate input value.

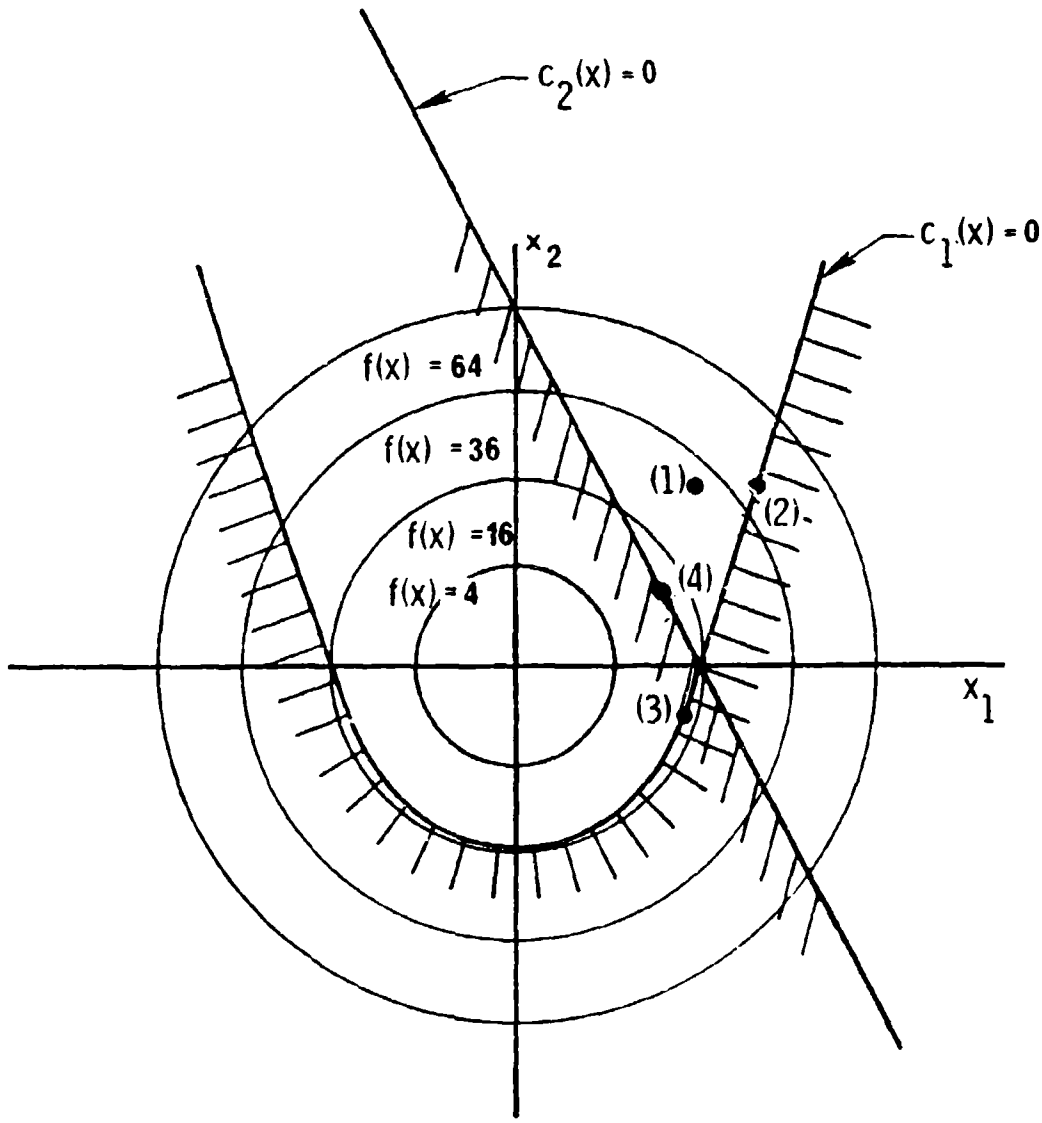


Figure 2.2-1. UOPTIM Example

Mnemonic	Description	Default Value
PENLTY	Value of the penalty parameter	1.
ITC	Problem number for analytic test problem (integer)	0
MAXITC	Iteration limit for the constraint solving process	99
MAXITS	Iteration limit for the unconstrained optimization process	99
MAXITP	Limit on the number of equality constrained problems that may be posed	20

2.2.1.6 Interpretive Output from UOPTIM

The following set of messages appear during the normal operation of the algorithm and are designed to aid the user in following the progress of the algorithm. These messages are printed in addition to the point-by-point iteration print described in Section 2.3.

OBJECTIVE FUNCTION + PENALTY = xxxxx... the value of f(x) (Eq. (4)).

DEFINE EQUALITY CONSTRAINED PROBLEM NJMBER n... (see Section 2.2.1.2).

FOR THIS SUBPROBLEM THE VALUE OF THE PENALTY PARAMETER IS xxxxx... (see Eq. (4)).

BEGIN EQUALITY CONSTRAINED OPTIMIZATION ALGORITHM IN ORDER TO OPTIMIZE THE OBJECTIVE FUNCTION SUBJECT TO THE CONSTRAINTS IN THE CURRENT BASIS. (see Section 2.2.1.2)

CONSTRAINT SOLVING VARIABLES... a list of the independent variables specified by the factorization to solve the constraints. (Section 2.2.1.2)

OPTIMIZATION VARIABLES... a list of the independent variables specified by the factorization to optimize the objective function. (Section 2.2.1.2)

BEGIN CONSTRAINT SOLVING ALGORITHM. (Section 2.2.1.2)

CONSTRAINT SOLVING ALGORITHM ITERATION NUMBER n. (Section 2.2.1.2)

UNCONSTRAINED OPTIMIZATION ALGORITHM ITERATION NUMBER n. (Section 2.2.1.2)

END CONSTRAINT SOLVING ALGORITHM. THIS POINT SATISFIES THE CONSTRAINTS IN THE BASIS. (Section 2.2.1.2)

END UNCONSTRAINED OPTIMIZATION ALGORITHM. THIS POINT OPTIMIZES THE OBJECTIVE FUNCTION SUBJECT TO THE CONSTRAINTS IN THE BASIS.

NO CONSTRAINT CAN BE DELETED FROM THE CURRENT BASIS (CASE 1) . . . constraint deletion logic to attempt to form a new basis.

NO CONSTRAINT CAN BE DELETED FROM THE CURRENT BASIS (CASE 2). . . constraint deletion logic to attempt to form a new basis.

THE FOLLOWING INEQUALITY CONSTRAINT HAS BEEN RELEASED FROM THE CURRENT BASIS (CASE 1) n. . . constraint deletion logic to attempt to form a new basis.

THE FOLLOWING INEQUALITY CONSTRAINT HAS BEEN RELEASED FROM THE CURRENT BASIS (CASE 2) n. . . constraint deletion logic to attempt to form a new basis.

THE FOLLOWING n VIOLATED INEQUALITY CONSTRAINTS HAVE BEEN ADDED TO THE CURRENT BASIS. . . a list of the constraints added to form a new basis. (see Section 2.2.1.3)

THE BASIS CONSISTS OF THE FOLLOWING n CONSTRAINTS. . . a list of constraints in the current basis. (see Section 2.2.1.2)

2.2.1.7 Diagnostic Output

Any one of the following set of messages may appear if the UOPTIM algorithm is not progressing in a nominal manner. The appearance of one or more of these messages should serve as an indication of a possible problem; however, it does not necessarily imply that the algorithm will be unable to obtain a solution.

IN THE CONSTRAINT SOLVING ALGORITHM THE CURRENT POINT IS ON THE BOUNDARY AND THE CONSTRAINTS ARE NOT SATISFIED. The constraint solving algorithm has terminated on a bound defined by Eq. (4). This

condition may be an indication of an unrealistic specification of the bounds or an inadequately defined problem.

REQUEST A NEW SET OF CONSTRAINT SOLVING VARIABLES. In an attempt to solve the constraints in the basis, a new set of constraint solving variables is requested.

THE CONSTRAINTS IN THE BASIS CANNOT BE SATISFIED WITH THE CURRENT SET OF CONSTRAINT SOLVING VARIABLES. The constraint solving algorithm is unable to satisfy the constraints. The algorithm will attempt to formulate a new problem by changing the constraint solving variables or by altering the basis.

IN THE OPTIMIZATION ALGORITHM THE CURRENT POINT IS ON THE BOUNDARY AND THE ALGORITHM CAN DO NO MORE PROCESSING. The unconstrained optimization algorithm has terminated on a bound defined by Eq.(4). This condition may be an indication of an unrealistic specification of the bounds or an inadequately defined problem.

END EQUALITY CONSTRAINED OPTIMIZATION PROCESS. THE CONSTRAINTS IN THE BASIS ARE INCONSISTENT OR INDETERMINATE AND CANNOT BE SATISFIED. The algorithm will attempt to formulate a new basis by deleting a satisfied inequality constraint.

NO INEQUALITY CONSTRAINT IN THE BASIS IS SATISFIED. No equality constraint can be deleted. The algorithm will terminate.

IN AN ATTEMPT TO FIND A BASIS THAT CAN BE SATISFIED, THE FOLLOWING INEQUALITY CONSTRAINT IS BEING RELEASED FROM THE CURRENT BASIS n. If a solution to the constraints in the new basis is found, then the algorithm will continue as outlined in Sections 2.2.1.3 and 2.2.1.2.

IMPOSSIBLE INCONSISTENCY ERROR CONDITION. The constraints are probably inconsistent. This may indicate an ill-posed problem.

THE ALGORITHM TERMINATED ON A MAXIMUM NUMBER OF ITERATIONS COUNT AND MAY NOT HAVE CONVERGED TO A PROPER SOLUTION. Further iteration will probably not yield any further improvement. An alternate problem formulation may be desirable.

THE PROGRAM DETECTED A PROBABLE CYCLING BEHAVIOR IN THE BASIS SELECTION PROCESS. The basis selection process is unable to determine the basic set of constraints. An alternate problem formulation may be desirable.

A PROBABLE ESSENTIAL INCONSISTENCY IN THE CONSTRAINTS HAS BEEN ENCOUNTERED. The algorithm has determined that the constraints do not have a solution, at least locally. A different starting point or an alternate problem formulation may be desirable.

THIS POINT IS ON A BOUNDARY AND THE ALGORITHM MAY NOT HAVE CONVERGED TO A PROPER SOLUTION. The algorithm is unable to find a solution that is interior to the region defined by Eq. (4). A larger value of the input value of PENLTY, or an alternate problem formulation, may be desirable.

2.2.2 UBEST

The optimization operator UBEST is described in the following sections. The operator is designed to solve the following problem. Determine the n-vector x that optimizes (i.e., maximizes or minimizes) the scalar function

$$f(x) = f(x_1, \ldots, x_n) \tag{1}$$

subject to the equality constraints

$$c_i(x) = 0 \qquad i = 1, 2, \ldots, k \tag{2}$$

and the inequality constraints

$$c_i(x) \geq 0 \qquad i = k+1, k+2, \ldots, m \tag{3}$$

The functions $f(x)$ and $c_i(x)$ are assumed to be twice continuously differentiable in the region

$$x_L \leq x \leq x_U \tag{4}$$

where x_L and x_U are specified lower and upper bounds. These bounds determine a region of computability and unlike constraints cannot be violated during the iterative process.

As special cases of this problem, the UBEST operator is designed to solve unconstrained optimization problems in which there are no constraints ($m = 0$), nonlinear root solving problems, search problems, and nonlinear least squares problems.

The following sections give a brief, user-oriented description of the UBEST algorithm. A more complete development of the algorithm may be found in Refs. 2, 3, and 4.

2.2.2.1 Preliminary Developments

Define the Lagrangian function

$$L(x,\lambda) = f(x) + c^T(x)\lambda \tag{5}$$

where $c(x)$ is the m-vector of all constraints and λ is the m-vector of Lagrange multipliers. At the optimum point (x^*, λ^*)

$$\nabla L(x^*, \lambda^*) = g(x^*) + G(x^*)\lambda^* = 0 \tag{6}$$

where ∇L is the gradient of the Lagrangian function with respect to x, $g(x)$ is the gradient of the objective function, and the $n \times m$ Jacobian matrix is given by

$$G(x) = \left[\nabla c_1, \ldots, \nabla c_m\right] = \begin{bmatrix} \frac{\partial c_1}{\partial x_1} & \cdots & \frac{\partial c_m}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial c_1}{\partial x_n} & \cdots & \frac{\partial c_m}{\partial x_n} \end{bmatrix} \tag{7}$$

Furthermore, at the solution

$$\lambda_i^* c_i(x^*) = 0 \qquad i = 1, \ldots, m \tag{8}$$

where

$$\lambda_i^* \leq 0 \qquad \text{for} \qquad i = (k+1), \ldots, m \tag{9}$$

To distinguish the constraints that are satisfied as equality constraints at a solution, define the basic set of constraints,

$$B^* = \left\{ c_i \middle| c_i(x^*) = 0 \qquad i = 1, \ldots, m \right\} \tag{10}$$

An estimate of B^* shall be referred to as a basis, B. If the gradients of the constraints in B^* are linearly independent at x^*, then Eqs. (6), (8), and (9)

constitute the Kuhn-Tucker necessary conditions for the existence of an optimum.

Assume a basis estimate has been made. The basic philosophy used by the optimization operator UBEST is to find a point where the Lagrangian condition, Eq. (6), is satisfied and then to estimate a point where the constraints are satisfied. At such a point a new basis estimate can be made, and the process can be repeated. In contrast, for a given basis, the operator UOPTIM first satisfies the constraints and then follows the constraint surface to a point where the Lagrangian condition is met.

2.2.2.2 Unconstrained Optimization Algorithm

It can be demonstrated that the Lagrangian condition, Eq. (6), is satisfied at points which minimize the augmented penalty function.

$$J(x,\lambda,r) = f(x) + c^T(x)\lambda + rP(x) = L(x,\lambda) + rP(x) \qquad (11)$$

where r is a scalar referred to as the penalty weight, λ is an m-vector of specified estimates of the Lagrange multipliers, and $P(x)$ is the penalty function

$$P(x) = \sum_{i \in B} c_i^2(x) + \sum_{i \notin B} I[c_i(x)]\, c_i^2(x) \qquad (12)$$

with the indicator I defined by

$$I[c_i(x)] = \begin{cases} 1 & \text{if } c_i < 0 \\ 0 & \text{if } c_i \geq 0 \end{cases} \qquad i = 1, 2, \ldots, m \qquad (13)$$

For specified values of λ and r, the minimization of J is an unconstrained optimization problem.

The basic procedure used to solve this unconstrained problem is to make a series of iterative improvements to an initial estimate of the solution, x^0, according to the formula

$$x^{i+1} = x^i - \rho^i s^i \qquad i = 0, 1, 2, \ldots \qquad (14)$$

The search direction s^i is computed by solving the augmented system

$$\begin{pmatrix} H^i & \hat{G} \\ \hat{G}^T & -\frac{1}{2r} I \end{pmatrix} \begin{pmatrix} s^i \\ \hat{\lambda} \end{pmatrix} = \begin{pmatrix} \nabla J(x^i) \\ 0 \end{pmatrix} \tag{15}$$

where ∇J is the gradient of J, $\hat{G}$ is the Jacobian matrix of the active constraints, and an i-th estimate of the Hessian matrix or matrix of second partial derivatives of J is defined by $\nabla^2 J = H^i + 2r\hat{G}\hat{G}^T$. The scalar ρ^i is chosen using a cubic search procedure so that

$$J(x^{i+1}) < J(x^i)$$

At the point x^{i+1}, a new estimate of the Hessian matrix can be constructed based on the values H^i, $J(x^{i+1})$, $J(x^i)$, $\nabla J(x^{i+1})$, and $\nabla J(x^i)$. This procedure can then be repeated to obtain a new estimate of the solution. If the search direction vector computed by Eq. (15) violates the bounds, Eq. (4), a quadratic programming algorithm is employed to compute s^i. This iterative process continues until

$$\| \nabla J(x^i) \| \leq \delta_1 \tag{16}$$

or

$$\sigma_r = \| x^{i+1} - x^i \| (\| x^i \| + 1)^{-1} \leq \delta_2 \tag{17}$$

where δ_1 and δ_2 are specified tolerances on the gradient norm and the resolution norm. If J is a quadratic function of the independent variables x, convergence will occur after n-iterations, i.e., the process is quasi-second order.

2.2.2.3 Determination of the Basis

The unconstrained optimization algorithm is applicable for a fixed basis and corresponding fixed values for the Lagrange multipliers. Estimates of the Lagrange multipliers are determined at a fixed point x to be the values of λ which minimize the error in Kuhn-Tucker conditions

$$e(\lambda) = \|A\lambda + b\|^2 \tag{18}$$

where

$$A = \begin{bmatrix} G \\ \hline c_1 & 0 & \cdots & 0 \\ \vdots & c & \ddots & \vdots \\ 0 & & \ddots & c_m \end{bmatrix} \qquad b = \begin{bmatrix} g \\ \hline 0 \end{bmatrix} \tag{19}$$

The minimization of e as a function of λ, subject to the negativity constraints, Eq. (9), is solved using a quadratic programming algorithm.

The estimates of the multipliers and the values of the constraints are used to determine a new basis $\bar{B}$ from the old basis B according to the following rules:

Rule 1. If $i \leq k$, then $i \in \bar{B}$; (20)

Rule 2. If $i \in B$, then $i \in \bar{B}$ iff ($\lambda_i < 0$ or $c_i \leq \delta_1$) (21)

Rule 3. If $i \notin B$, then $i \in \bar{B}$ iff ($\lambda_i < 0$ and $c_i \leq \delta_1$) (22)

Rule 1 requires that all equality constraints be in the basis. Rule 2 gives the conditions for deleting a constraint; namely the i-th constraint is deleted if it is satisfied and has a nonnegative multiplier. According to Rule 3, a constraint is added if its multiplier is negative and the constraint is violated.

It is possible that there is no point which satisfies the constraints in which case the constraints are called inconsistent. If there is an indication of constraint inconsistency, then $\bar{B}$ is constructed from B by deleting all satisfied inequality constraints.

2.2.2.4 Constraint Phase

The purpose of the constraint solving phase is to predict a point where the constraints are satisfied, while keeping the Lagrangian condition, Eq. (6), satisfied. An estimate of the basis is required and the previously

described unconstrained optimization algorithm is applied to the penalty function alone, ignoring the contribution of $L(x,\lambda)$ to $J(x,\lambda,r)$. The procedure assumes the objective function is quadratic and the constraints are linear. In particular, a direction is obtained by solving Eq. (15) in its limiting form

$$\begin{pmatrix} H^i & \hat{G} \\ \hat{G}^T & 0 \end{pmatrix} \begin{pmatrix} s^i \\ \hat{\lambda} \end{pmatrix} = \begin{pmatrix} g^i \\ \hat{c}^i \end{pmatrix} \tag{23}$$

where $\hat{c}$ denotes the $\hat{m}$ constraints in the current basis and any other violated constraints, $\hat{G}$ denotes the Jacobian matrix of the active constraints, and $\hat{\lambda}$ denotes the corresponding multipliers. The search step is then given by

$$\bar{x} = x - \beta s^i \tag{24}$$

where the scalar step length β is chosen such that $P(\bar{x}) < P(x)$. Nominally $\beta = 1$, and a cubic interpolation procedure is used if necessary.

At the end of the constraint phase, the penalty weight is increased according to the computation

$$r^{k+1} = \max \left[\frac{-\nabla L^T \nabla P}{\nabla P^T \nabla P} \quad , \quad 10 \max_i (\lambda_i^2) \, , \, \kappa r^k \right] \tag{25}$$

where κ is an input quantity. When the basis is unchanged from the Lagrangian phase to the constraint phase, κ is set to one, i.e., there is no increase in the penalty weight.

2.2.2.5 The Nonlinear Programming Algorithm

The basic steps of the algorithm are:

Step 1. (Lagrangian Phase) For a fixed basis B^k, fixed multipliers λ^k, and fixed penalty weight r^k, minimize the augmented penalty function, Eq. (11), using the unconstrained optimization algorithm given in Section 2.2.2.2. Call the solution $\bar{x}$.

Step 2. (Basis Determination) Keeping $\bar{x}$ fixed, compute a new

basis $\bar{B}$ and multipliers $\bar{\lambda}$, using the procedure in Section 2.2.2.3.

Step 3. (Constraint Phase) Beginning at $\bar{x}$, with fixed basis $\bar{B}$, minimize P, using the unconstrained algorithm given in Section 2.2.2.4. Call the solution x^{k+1}. If $P(x^{k+1}) \neq 0$, constraints may be inconsistent.

Step 4. (Basis Determination) Keeping x^{k+1} fixed, determine a new basis B^{k+1} and multipliers λ^{k+1} using the procedure given in Section 2.2.2.3. When checking for inconsistent constraints, if $\bar{B} = B^{k+1}$, terminate the algorithm.

Step 5. (Convergence Tests) Terminate if $e(\lambda) < \delta_1$ and $\sigma_r < \delta_2$ from Eqs. (16) and (17).

Step 6. (Define Penalty Weight) Compute new penalty weight using Eq. (25).

Step 7. (Update Information) Set $k = k+1$, $x^k = x^{k+1}$, $r^k = r^{k+1}$, $\lambda^k = \lambda^{k+1}$, $B^k = B^{k+1}$, etc. Return to Step 1.

2.2.2.6 Example

As an illustration of the behavior of the algorithm, consider the following example:

$$\text{minimize } f(x) = 100(x_2 - x_1^2)^2 + (1-x_1)^2$$

subject to

$$c_1(x) = x_2^2 + x_1 \geq 0$$

$$c_2(x) = x_1^2 + x_2 \geq 0$$

$$c_3(x) = -x_1 + \frac{1}{2} \geq 0$$

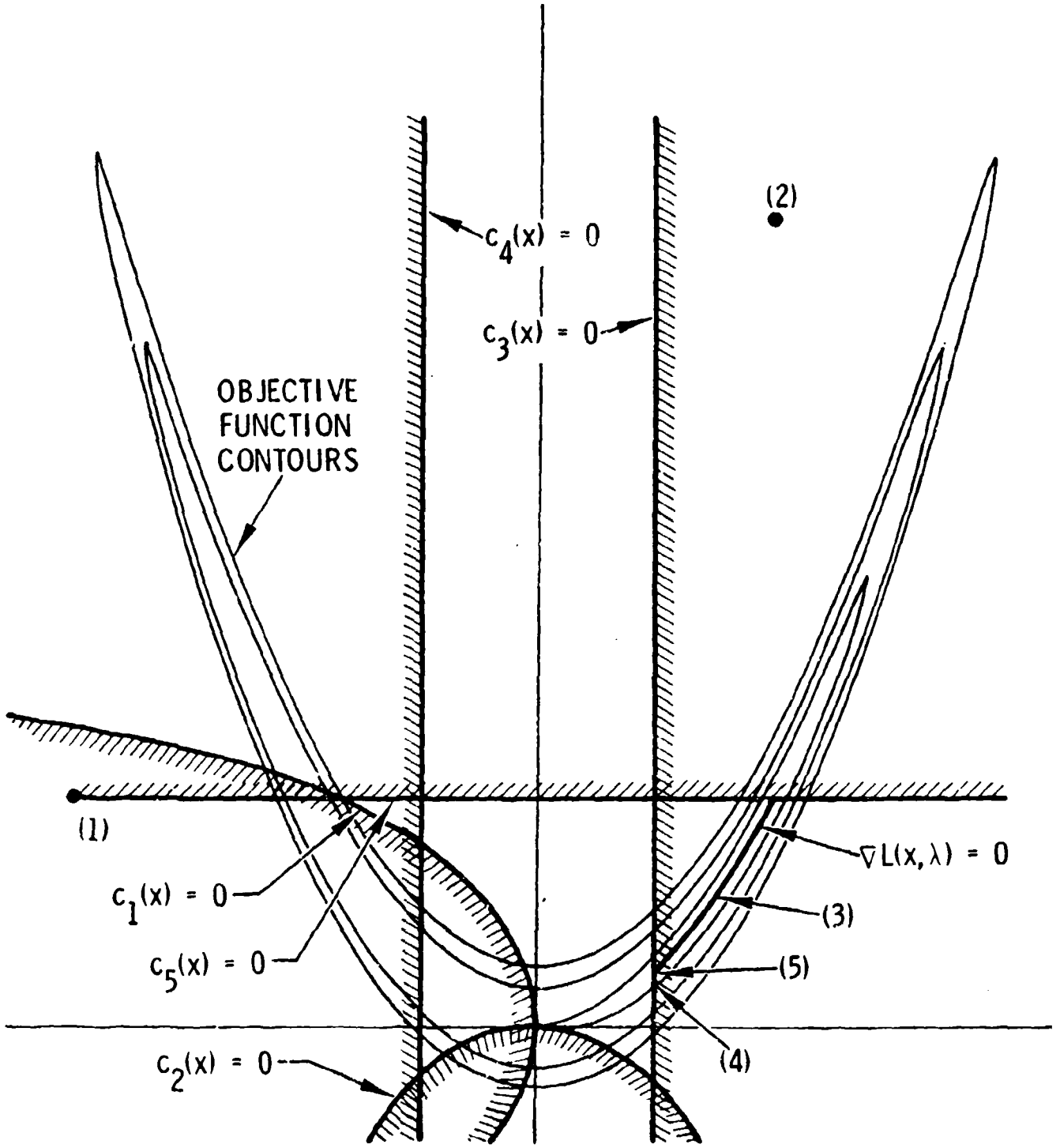


Figure 2.2-2. UBEST Example

$$c_4(x) = x_1 + \frac{1}{2} \geq 0$$

$$c_5(x) = -x_2 + 1 \geq 0$$

Figure 2.2-2 illustrates the contours of the objective function as well as the constraint boundaries. The violated regions for the constraints have been crosshatched. For this example there were no constraints in the initial basis, so the initial multipliers were set to zero. A penalty weight of $r^1 = 1$ was used and at the initial point, $x^0 = (-2, 1)$; the value of the augmented index $J(x^0, \lambda, r) = 915.25$. The first iteration of the unconstrained minimization produced the point (1.027, 3.614), which is labeled (2) in the Figure 2.2-2 and at this point $J = 662.03$. After 32 function evaluations (27 iterations), the unconstrained minimum of J was located at $x^1 = (0.75, 0.5625)$, labeled (3), completing Step 1 of the algorithm.

Observe that for this problem the Lagrangian condition $\nabla L = 0$, is satisfied when $x_2 - x_1^2 = 0$, for $0.5 \leq x_1 \leq 1$, and clearly x^1 lies on this curve. The basis was determined as described in Step 2. Constraint c_3 was added to the basis, with the multiplier estimate $\lambda_3 = -0.4705$. Beginning at x^1, an estimate of the constrained solution was made, assuming c_3 was in the final basis. The point $\bar{x} = (0.5, 0.1873)$ is shown with a (4) on the figure. Note that the point satisfies the constraint, since c_3 is linear. After making a new estimate of the basis in Step 4, and increasing the penalty weight in Step 6, the algorithm returns to Step 1, with $r^2 = 10$. The unconstrained minimum of J was located at $x^2 = (0.545, 0.297)$ labeled (5). Two more constraint phases and two more unconstrained minimizations were performed before the solution at $x^* = (0.5, 0.25)$ was obtained. A total of 47 function evaluations were required to guarantee five place accuracy.

2.2.2.7 Special Usage

The following quantities which are specific to the operator UBEST may be specified as a part of the problem definition model, PRBDFM1 input. (See Section 2.3.2.1.)

However, it is recommended that these quantities not be changed from their preset values without knowledge of the program and a demonstrated requirement for an alternate input value.

Mnemonic	Description	Default Value
DELTA1	dependent variables convergence tolerance (see Step 5, and Eq. (16)); i.e., acceptable error in Kunn-Tucker conditions.	10^{-3}
DELTA2	independent variable resolution convergence tolerance (see Step 5, and Eq. (17)); guarantees $\log \frac{1}{\delta_2}$ significant figures accuracy in the independent variables.	10^{-3}
CONLIM	convergence limit.	10^{-10}
EUSE	typically EUSE>DELTA2, determines when to use the U matrix in the Hessian computations. (see Ref. 3)	10^{-2}
ITMAX	the number of improving steps taken in an unconstrained search iteration. An improved step decreases the augmented objective function.	1
PERSNT	percentage of the range of the variables to be permitted on first search evaluation.	0.01
ZERO	floating point numbers whose absolute values are less than ZERO treated as 0.	10^{-12}
DELTR	κ in Eq. (25), smallest penalty increment per cycle.	10
PENLTY	the scalar r in Eq. (11). If PENLTY $\leq$ 0, a	1

Mnemonic	Description	Default Value
	basis estimate is made at the initial point and r = max (|PENLTY|, 1.).	
MODEOP	Mode of operation. When MODEOP = 1, begin with Lagrangian phase (Step 1). When MODEOP = 2, begin with constraint phase (Step 3). When MODEOP = 3, begin with Step 4.	1

2.2.2.8 Interpretive Output from UBEST

The following set of messages appear in the normal operation of the algorithm and are designed to aid the user in following the progress of the algorithm. These messages are printed in addition to the point by point iteration print described in Section 2.3.

OBJECTIVE FUNCTION SCALE WEIGHT . . . the objective function scale factor.

CONSTRAINT SCALE WEIGHTS . . . the constraint scale factors.

VARIABLE SCALE WEIGHTS . . . independent variable scale factors.

CONVERGENCE TOLERANCES . . . . . DELTA1 = ___, DELTA2 = ___. Eqs. (16) and (17)

ITERATION NUMBER ___, GRADIENT NORM = ___, AUGMENTED INDEX = ___, ... display $\|\nabla J\|$ and J. (Section 2)

SEARCH . . . STEP = ___ INDEX REDUCTION = ___ . . . display one-dimensional search information, ρ and $J(x^{i+1}) - J(x^i)$. (Section 2.2.2.2)

CONVERGENCE . . . . . GRADNM = ___, RESNOR = ___,
SOLUTION HAS BEEN OBTAINED WITH R = ___. . . convergence of unconstrained algorithm, display $\|\nabla J\|$, σ_r, and r. (Section 2.2.2.2)

SEARCH VECTOR COMPUTED USING Q. P. ALGORITHM . . . quadratic programming algorithm was used to compute the s vector. (Section 2.2.2.2)

U MATRIX USED FOR ALL SUBSEQUENT ITERATIONS . . . U matrix used in the Hessian matrix approximation. (see Ref. 2)

LEAST-SQUARES SOLUTION OBTAINED . . . unconstrained optimization algorithm used to minimize penalty only, i.e., J = P. (Section 2.2.2.2)

CONSTRAINTS IN BASIS . . . display basis constraint indices in order of violation. (Section 2.2.2.3)

LAGRANGE MULTIPLIERS . . . display estimates of Lagrange multipliers, λ_i. (Section 2.2.2.3)

ERROR IN GRADIENT . . . the error in the Lagrangian condition, $\|\nabla L(x^*,\lambda^*)\|$ in Eq. (6).

ERROR IN CONSTRAINTS . . . the error in the constraints, $\left[P(x^*)\right]^{1/2}$.

INDEPENDENT VARIABLE RESOLUTION ERROR . . . error in the variables, x and λ.

2.2.2.9 Diagnostic Output from UBEST

The following list contains messages which may appear if the algorithm is behaving in an abnormal fashion. The appearance of one or more of these messages should serve as an indication of a possible problem. It does not

necessarily imply that the algorithm will be unable to terminate normally.

CONVERGENCE TOLERANCES CHANGED . . . . DELTA1 = ___. DELTA2 = ___Internal convergence tolerances changed, usually because one but not both of the input tolerances are satisfied. The input convergence tolerances may be inconsistent or unrealistic.

PSEUDORANK = ___ LESS THAN ___. . . Possible numerical difficulties in the linear least squares process used to compute the pseudoinverse. The problem may be poorly scaled.

SOLUTION MAY BE A DEGENERATE LOCAL MINIMUM . . . Possible numerical difficulties in the quadratic programming algorithm. The problem may be poorly scaled. This message often occurs in conjunction with the previous message.

HESTENES ESTIMATE OF MULTIPLIERS USED . . . Numerical difficulties computing the Lagrange multipliers. The approximation $\lambda_i = 2rc_iI[c_i]$ is used.

CONSTRAINTS MAY BE LINEARLY DEPENDENT . . . The gradients of the constraints may not be linearly independent at the solution, in which case the Kuhn-Tucker conditions are not applicable. One may expect deterioration in the performance of the algorithm, as well as other indications of numerical problems. An alternate formulation of the problem may be appropriate.

THE FOLLOWING CONSTRAINTS ARE INCONSISTENT . . . The penalty function has a nonzero value at its minimum. No constraint could be deleted from the current basis. There may not be a point which satisfies the constraints, in which case the problem should be reformulated. The constraints may be locally inconsistent, in which case restarting the algorithm from the initial point with a larger penalty weight may lead to a solution.

NUMBER OF EQUALITY CONSTRAINTS = ___, GREATER THAN N. Unless this is a search problem (PROB = :SEARCH:), it may be poorly formulated.

TERMINATION BECAUSE POINT IS ON BOUNDARY . . . the unconstrained algorithm has terminated with one of the variables on the boundary. The point may not be on the $\nabla L = 0$ surface. This may imply a poor problem formulation and/or incorrect use of the bounds. The solution may be nonoptimal.

2.2.3 USCHN

The search operator USCHN is described in the following sections. This operator determines the values of a set of n variables

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix} \tag{1}$$

which satisfy the m equality constraints

$$c_i(x) = 0 \quad i = 1, 2, \ldots, m \tag{2}$$

where $m \leq n$. The region of search is limited by

$$x_L \leq x \leq x_U \tag{3}$$

where x_L and x_U are specified lower and upper bound vectors. These bounds determine a region of computability and, unlike constraints, cannot be violated during the iteration process.

2.2.3.1 Basic Procedure

The user specifies an initial point $x^{(0)}$. In addition, he must specify a perturbation step size Δx_j for each of the n variables.

Nominally, an initial difference matrix $D^{(0)}$ is computed at the initial point by

$$D_{ij}^{(0)} = c_i(x^{(0)}) - c_i(x^{(0)} - \Delta_j x) \qquad i = 1, \ldots, m;\ j = 1, \ldots, n \tag{4}$$

where $\Delta_j x$ is a vector whose components are zero except for the j-th component which is Δx_j. Optionally, $D^{(0)}$ may be specified by the user or may be generated in the previous case.

USCHN generates an estimate $x^{(1)}$ of the solution by

$$x^{(1)} = x^{(0)} - \delta x \tag{5}$$

where δx is to be determined. Two requirements are enforced on $x^{(1)}$. First, $x^{(1)}$ must be within the region defined by Eq. (3). Secondly, the change in any of the variables in going from $x^{(0)}$ to $x^{(1)}$ must be no more than a multiple $\eta^{(0)}$ of the input perturbation size, where $\eta^{(0)}$ is a scalar. That is, δx is required to satisfy

$$x_L \leq x^{(0)} - \delta x \leq x_U \tag{6}$$

and

$$|(\delta x)_j| \leq \eta^{(0)} \Delta x_j \qquad j = 1, 2, \ldots, n \tag{7}$$

USCHN accomplishes this in the following manner. Define a trial bounding vector β_t by

$$(\beta_t)_j = \min \{ \eta^{(0)} |\Delta x_j|, |x_j^{(0)} - (x_L)_j|, |(x_U)_j - x_j^{(0)}| \} \tag{8}$$

$$j = 1, 2, \ldots, n$$

Define the diagonal matrix ΔX by

$$(\Delta X)_{jj} = \Delta x_j \tag{9}$$

Let the n-dimensional vector p minimize

$$\| c(x^{(0)}) - D^{(0)} p \| \tag{10}$$

subject to the constraint

$$\sum_{j=1}^{n} \left(\frac{p_j \Delta X_j}{\beta_j} \right)^2 \leq 1 \tag{11}$$

Then δx defined by

$$\delta x = (\Delta X) p \tag{12}$$

satisfies (6) and (7), and we generate the trial point x_t by

$$x_t = x^{(0)} - (\Delta X) p \tag{13}$$

We comment at this point that if the matrix J^* is defined by

$$J^* = D^{(0)} (\Delta X)^{-1}$$

then

$$(J^*)_{ij} = \frac{c_i(x^{(0)}) - c_i(x^{(0)} - \Delta_j x)}{\Delta x_j}$$

Thus, $J^* = D^{(0)}(\Delta X)^{-1}$ is an approximation to the Jacobian matrix $J = (\partial c / \partial x)$. Equations (10), (12), and (13) are then seen to represent a Newton-like method for determining a solution to the set of constraints using J^* as an approximation to the Jacobian matrix. The restriction Eq. (11) ensures that Eqs. (6) and (7) are satisfied.

Before the trial point x_t is accepted as the point $x^{(1)}$, the algorithm must determine whether "sufficient improvement" in the constraints has been obtained at x_t. The method for this decision is explained in Section 2.2.3.2. If sufficient improvement has been obtained, x_t is accepted as the new point $x^{(1)}$ and the difference matrix $D^{(0)}$ is updated by

$$D^{(1)} = D^{(0)} - \Delta D^{(1)} \tag{14}$$

where the matrix $\Delta D^{(1)}$ is computed by

$$(\Delta D^{(1)})_{ij} = \frac{c_i(x^{(1)})\, p_j}{\|p\|^2} \qquad i = 1, \ldots, m;\ j = 1, \ldots, n \tag{15}$$

In addition, an $\eta^{(1)}$ is computed from $\eta^{(0)}$ and the amount of gain. This procedure is repeated until a solution is found or an unsatisfactory condition occurs.

Hence, the general iteration scheme for the USCHN algorithm is as follows. Assume $x^{(k)}$, $c(x^{(k)})$, $D^{(k)}$, and $\eta^{(k)}$ have been determined. Calculate the solution p which minimizes

$$\|c(x^{(k)}) - D^{(k)}p\| \tag{16}$$

subject to the constraint

$$\sum_{j=1}^{n} \left(\frac{p_j \Delta X_j}{\beta_j^{(k)}} \right)^2 \leq 1 \tag{17}$$

where

$$\beta_j^{(k)} = \min \{ \eta^{(k)} \Delta x_j, |x_j^{(k)} - (x_L)_j|, |(x_U)_j - x_j^{(k)}| \} \tag{18}$$

$$j = 1, \ldots, n$$

Compute the trial point

$$x_t = x^{(k)} - (\Delta x)p \tag{19}$$

If "sufficient improvement" in $c(x_t)$ is achieved as defined by Section 2.2.3.2, x_t is accepted as the new point $x^{(k+1)}$. The matrix $D^{(k)}$ is updated by

$$D^{(k+1)} = D^{(k)} - \Delta D^{(k+1)} \tag{20}$$

where

$$(\Delta D^{(k+1)})_{ij} = \frac{c_i(x^{(k+1)})\, p_j}{\|p\|^2} \tag{21}$$

and the scalar $\eta^{(k)}$ is updated to $\eta^{(k+1)}$ by determining how much gain in the constraints has been made.

This procedure is continued until either the convergence criteria has been satisfied or a maximum number of predictions has been made. The convergence criteria is that

$$|c_i(x^{(k)})| < \epsilon_i \qquad i = 1, 2, \ldots, m \tag{22}$$

where the tolerances ϵ_i are specified by the user. Currently, no resolution test is made on the independent variables.

2.2.3.2 Update Logic

The purpose of the update logic is to determine if sufficient improvement has been obtained at a trial point. Also, if sufficient improvement has not been obtained, then the update logic specifies a procedure for generating a new trial point. Specifically, the gain g is defined to be the square root of the ratio of the sum of the squares of the constraints at a previous or reference point $x^{(k)}$ to the sum of the squares of constraints at a trial point x_t. That is

$$g = \sqrt{\frac{\sum_{i=1}^{m} (c_i(x^{(k)})^2}{\sum_{i=1}^{m} (c_i(x_t))^2}} \tag{23}$$

Clearly, the gain indicates if the USCHN algorithm is converging $(g > 1)$ or diverging $(g < 1)$.

If the gain g is greater than 1., sufficient improvement has been obtained and x_t is accepted as the new point $x^{(k+1)}$. Hence if $g \geq 1.$, then the matrix D is updated as in Eqs. (20) and (21), the trial point becomes the new reference point, and a new value of the boundary parameter $\eta^{(k+1)}$ is computed.

If $g < 1.$, then the matrix D is not updated. Instead, an attempt is made to generate a new trial point which will show sufficient improvement. The basic procedure for generating a new point is to reduce the bounding factor η and then repeat the computations given in Eqs. (16) through (19). The reference point remains unchanged and the new point is considered to be another trial point. The value of η is computed as a function of the gain g the number of trial points generated from the current reference point, and a prediction of the gain generated as a part of the least squares process.

2.2.3.3 USCHN Algorithm

The basic steps of the USCHN algorithm are:

Assume $D^{(k)}$, $x^{(k)}$, $\eta^{(k)}$ are given where $x^{(k)}$ is the current reference point.

Step (1). Compute the vector p by Eqs. (16) through (18) and set $x_t = x^{(k)} - (\Delta x)p$. Go to Step (2).

Step (2). Evaluate $c(x_t)$, and test for convergence. If the convergence criteria has been satisfied, terminate. Otherwise, go to Step (3).

Step (3). Compute the gain by Eq. (23). If sufficient improvement has been made to go to Step (4). Otherwise go to Step (5).

Step (4). Compute the updated difference matrix $D^{(k+1)}$ by Eqs. (20) and (21), compute $\eta^{(k+1)}$, and set $x^{(k+1)} = x_t$. Replace k by k+1, and go to Step (1).

Step (5). Restrict the boundary parameter η, and leave the current reference point unchanged. Go to Step (1) to generate a new trial point.

2.2.3.4 Interpretive Output from USCHN

The following set of messages appear in the normal operation of the algorithm and are designed to aid the user in following the progress of the algorithm. These messages are printed in addition to the point-by-point iteration print described in Section 2.4.

ON TRIAL i THE MAXIMUM INDEPENDENT VARIABLE STEP SIZE WAS xx.xxx PERTURBATION STEPS. . . . the limit on the magnitude of the vector p (Eqs. (16), (17) for the i-th trial).

THE ACTUAL STEP TAKEN WAS *** xx.xxx PERTURBATION STEPS. . . . the actual magnitude of the vector p, Eqs. (16), (17).

THE PREDICTED GAIN WAS *** xx.xxx. . . the predicted value of the gain, Eq. (23).

THE ACTUAL GAIN WAS **** xx.xxx. . . the computed value of the gain, Eq. (23).

LAST POINT BECOMES NEW REFERENCE POINT. UPDATE THE DIFFERENCE MATRIX. . . the difference matrix is updated, Eqs. (14), (15).

2.3 Optimization Input

The input data required to define an optimization or search problem is discussed in this section. All optimization input data is specified by the GTL input language which is described in Volume II. Accordingly, the description of the input data given here will use the notation and terminology of GTL defined in Volume II. The optimization data is input in the model type-model format. The model types reflect the major components of an optimization or search problem; namely, an overall problem definition model type, an objective function definition model type, a constraint definition model type, and an independent variable definition model type. These model types and the individual input models are discussed in succeeding sections. Furthermore, these models are compatible with any of the operators discussed in Section 2.2. Thus, the same constraint model, for example, may define constraints for UOPTIM, UBEST, or USCHN. Unless otherwise indicated, all input quantities are applicable to any operator. Before discussing the individual models, a summary of the optimization input format is presented.

2.3.1 Optimization Input Format

The optimization data is contained in a model type data block. The name of this data block is arbitrary, except that it must be a unique GTL symbol. The data within this data block is in the model type-model name format. The general format of the optimization data is the following:

```
                                    (6)
OPTDATA                    OPTIMIZATION INPUT DATA
                    (11)                (21)
                    PROBDEF             PRBDFM1     data and/or tables
                    OBJFTN              OBJFNM1     data and/or tables
                    CONSTR              CNSTRM1     data and/or tables
                    INDVAR              VARM1       data and/or tables
                      .
                      .
                      .
(1)
EXECUTE                 OPTSYS (OPTDATA)
```

The input statement EXECUTE OPTSYS (data block name) is required to execute the optimization program. The name within the parentheses must be the name of the data block which contains the input data that defines the optimization problem to be solved. This format permits several problems to be defined simultaneously by specifying a different data block name for each problem. The program only attempts to solve those problems which are referenced by having the corresponding data block name appear in an EXECUTE OPTSYS (data block name) statement.

For each model type, the optimization input data has the following format:

(1)	(2)	
Model type	Model name	data assignment statements /independent slash statement/ ((modified) expression statement) TABLE 1, TABLE 2 ... TABLE n

The data assignment stream must contain data appropriate for that model. For example, constraint model data must be input to a constraint model and not to a problem definition model.

TABLE 1, TABLE 2 ... TABLE n are general data tables. These tables must also contain data appropriate to the model which references the table. A requirement unique to the optimization input models is that all table names must appear at the end of the data assignment stream. That is, the first table name must follow all other data. For added flexibility in defining optimization problems, any of the data assignment statements, as well as the components of the independent slash statements or of the expression statements, may be equivalenced using the GTL equivalence option. The resulting user-defined symbol may then be used to specify input for the current problem or succeeding problems.

2.3.2 Problem Definition Models (PROBDEF)

Models associated with the model type PROBDEF define the general

characteristics of the problem to be solved and certain optimization process control options.

2.3.2.1 Problem Definition Model 1 (PRBDFM1)

Description

PRBDFM1 is an optimization input model which defines the overall problem to be solved. The following types of problems may be defined by this input model: optimization problems, search problems, root problems, and least squares problems. For optimization problems either maximization or minimization problems may be specified.

For search problems, the objective is to satisfy a set of equality constraints. Consequently, all constraints defined for a search problem are assumed to be equality constraints. The number of independent variables may be greater than or equal to the number of constraints. In general, if the number of independent variables is greater than the number of constraints, then the search problem may have many local solutions. For such problems the user then has the option of accepting the first point the algorithm determines which satisfies the constraints,or having the algorithm establish a further criterion to obtain a unique solution.

A restricted type of search problem is a root problem. For root solving problems the number of independent variables equals the number of constraints, and a unique point satisfies the constraints. All constraints defined for root solving problems are assumed to be equality constraints.

A least squares problem is actually an optimization problem. PRBDFM1 poses such a problem as a minimization problem for which the objective function is the sum of squares of a set of constraints (residuals). The residuals are defined as equality constraints. The objective is to determine the values of the independent variables which provide a best approximation to the solution of the constraints in the sense that the sum of the squares of the constraints (residuals) is minimized.

All data required to exercise the restart capability (see Section 2.1.4)

should also be specified as a part of the data input to PRBDFM1. For the current restart capability, the problem formulation must remain unchanged from the original problem specification for the subsequent restart. For example, the constraints and even the constraint tolerance must remain the same for a restart. Furthermore, a problem can only be restarted from the last point of a preceding run for which information was saved.

Model Inputs

The following two independent slash statements are required inputs to PRBDFM1:

/THE OPTIMIZATION OPERATOR IS optimization operator/

Currently, the optimization operator must be UOPTIM, USCHN, or UBEST.

/THE FUNCTION GENERATOR IS model name data /

The optional input "data" may be a data assignment stream or a data block name. Within these two slash statements, neither the optimization operator nor the model name of function generator may be equivalenced quantities.

The following quantities are input via data assignment statements. These quantities need to be specified only if the default values are not desired.

Mnemonic	Description	Default Value
PROB	Type of problem	MIN
	PROB = :MAX: - maximize the objection function (not applicable for USCHN)	
	PROB = :MIN: - minimize the objection function (not applicable for USCHN)	
	PROB = :SEARCH: - search problem	
	PROB = :ROOT: - root solving problem	
	PROB = :LSTSQ: - least squares problem (not applicable for USCHN)	
MAXCON	Maximum number of constraints (integer)	25

Mnemonic	Description	Default Value
MAXDIM	Maximum number of independent variables (integer)	15
MAXNFE	Maximum number of function evaluations. Algorithm will terminate if more than MAXNFE function evaluations are requested even though a solution has not been obtained (integer).	200
IOPTER	Function generator error response logic flag. (See Section 2.1.3) IOPTER = 0, the logic outlined in Section 2.1.3 is followed. IOPTER = 1, the job is terminated if the function generator is unable to complete the function evaluation at any point. (integer)	0
SRCHOP	Search option (applicable only to UOPTIM with PROB = :SEARCH:) SRCHOP = 1 - the first point which satisfies the constraints is retained as the solution. SRCHOP = 2 - the operator will determine a generalized inverse solution.	1
PRTMAX	A maximum allowable perturbation size. May be specified to limit the perturbation sizes.	1000.
PRTMIN	A minimum allowable perturbation size. May be specified to limit the perturbation sizes.	.-10

The following input quantities permit the program to sa... ...on

for a subsequent restart or to restart using previously saved information. Note that whenever information is stored,or saved information is accessed, then corresponding control cards are required.

Mnemonic	Description	Default Value
ISTORE	Flag to indicate information is to be stored for a subsequent restart. (integer) ISTORE = 0, no information is stored. ISTORE = 1, then information is stored only prior to function evaluations which begin near the central processor (CP) time limit of the submitted job. Specifically, information is saved only prior to function evaluations which occur in the CP time interval (TIMLMT - TSAVLST, TIMLMT - TSAVDEL). The parameters TIMLMT, TSAVLST, TSAVDEL are defined below. Nominally these parameters are computed by the storage algorithm and need not be input by the user. For special purpose applications, however, any one or all of these parameters may be input. Those which are not input are again computed by the algorithm.	0
TIMLMT	A parameter which determines if information is to be stored (see ISTORE). TIMLMT should equal the time limit of the job in decimal seconds. TIMLMT is applicable only if ISTORE = 1. The units of TIMLMT are seconds.	Computed by the storage algorithm
TSAVLST	A parameter which determines if information is to be stored (see ISTORE). TSAVLST should be larger than the computation time required to complete one function evaluation. TSAVLST	Computed by the storage algorithm

Mnemonic	Description	Default Value
	is applicable only if ISTORE = 1. The units of TSAVLST are seconds.	
TSAVDEL	A parameter which determines if information is to be stored. TSAVDEL may be input as a positive quantity to insure that the time limit does not occur during the storing of information. Thus, TSAVDEL should be larger than the time required to store the restart information. TSAVDEL is applicable only if ISTORE = 1. The units of TSAVDEL are seconds.	Computed by the storage algorithm
IRSTART	Restart flag. (integer) IRSTART = 0, this problem is not a restart of a previous problem. IRSTART = 1, this problem is a restart of a previous problem.	0
NOPSTOR	File number on which restart information is stored (integer). If information is stored, then file NOPSTOR must be saved in some manner. One method is the following control card generator option - 0,402 (TAPE25, TAPE=SAVE) (assuming the default value of NOPSTOR is used).	25
NOPRESR	File number from which saved information is to be read if a restart is requested (integer). If the information on file NOPRESR is to be accessed, then this file must be made available to the program in some manner. One method is the control card generator option -	26

Mnemonic	Description	Default Value
	0.403 (TAPE26, TAPE=tape number) (assuming the default value of NOPRESR is used).	

The following independent slash statements are listed for completeness, but they should not be changed from the default values without detailed knowledge of the program.

/MODEL TYPE MSIEVAL EXECUTES MODEL model name/
/MODEL TYPE MSIEEND EXECUTES MODEL model name/

2.3.3 Objective Function Models (OBJFTN)

Models associated with the model type OBJFTN define the objective function. If an objective function is not to be defined (e.g., search problem), then the OBJFTN model type should not be specified.

2.3.3.1 Objective Function Model 1 (OBJFNM1)

Description

OBJFNM1 is an optimization input model which defines the objective function for an optimization problem. For search problems, root solving problems, and least squares problems, there is no explicit objective function; consequently, this objective function model should not be specified.

Model Inputs

The objective function must be specified with a GTL expression statement in the following format:

$$(\text{function name}\left[\left|\begin{matrix}\text{.AT.}\\ \text{.OF.}\end{matrix}\right|\ \text{qualifier name}\quad\text{units}\right])$$

The function name is a required input. A qualifier is required only if further definition of the function name is required. For trajectory optimization problems with the partial trajectory option, the .AT. qualifier is required if "function name" is to be evaluated at an event other than the final event.

The input "units" is an optional input which specifies the external units for the objective function. This input is required only if the external units of the objective function are required to be different than the preset units of the objective function.

Associated with the expression statement given above, the following two optional slash statement modifiers are available to further specify the objective function.

/function name [.AT. / .OF.] qualifier] IS COMPUTED

BY routine name [data] /

This input is required only if the objective function is not computed by the normal execution of the function generator.

/function name [.AT. / .OF.] qualifier] PARTIALS ARE

COMPUTED ANALYTICALLY BY routine name [data] /

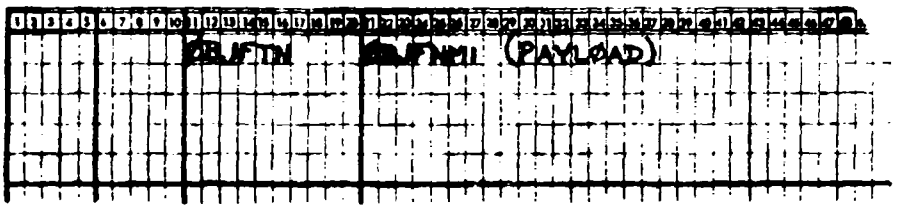
This input is required only if the partial derivatives of the objective function with respect to the independent variables are to be computed analytically by the specified routine. For both of these slash statement modifiers, the reference to the objective function must include any subscripts and/or qualifier given as a part of the definition of the objective function in the original expression statement.

Objective function scaling is specified by the following data assignment statement.

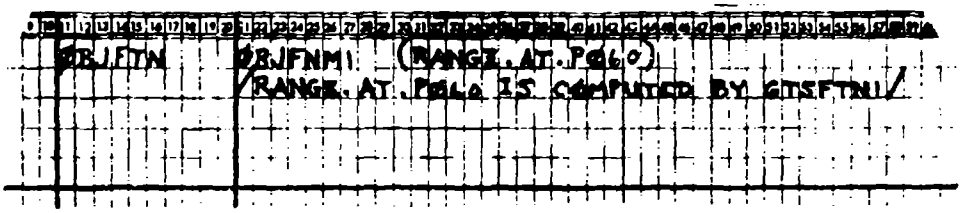
Mnemonic	Description	Preset
OBJSKL	OBJSKL = :UNSCALED: implies no objective function scaling	:UNSCALED:
	OBJSKL = :SCALED: implies the objective function is scaled and the scale weight is computed internally	
	OBJSKL = n.n implies the objective function is scaled and the scale weight is n.n	

Examples of OBJFNM1 input

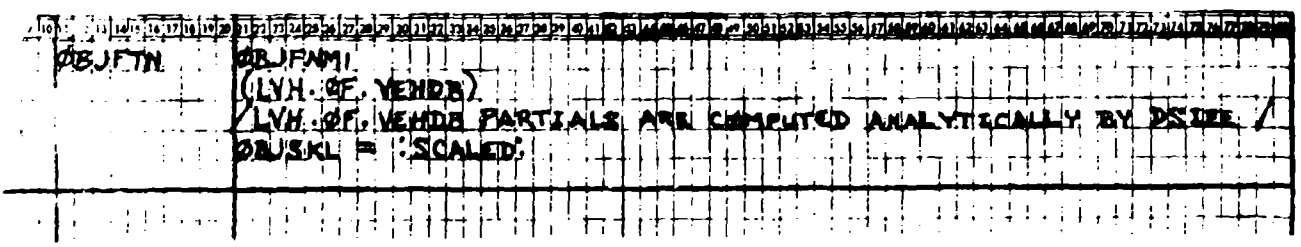
Example 1. For a trajectory optimization problem, assume the objective function is PAYLOAD which is not a time dependent quantity.



Example 2. For a trajectory optimization problem, assume the objective function is the variable RANGZ (evaluated at event PO60) which is computed by subroutine GTSFTN1. Furthermore, assume the partial trajectory option is specified.



Example 3. For a vehicle design problem, assume the objective function is the length of the total vehicle. Furthermore, assume objective function scaling is desired and the partial derivatives are computed analytically by the routine DSIZE. It is necessary to qualify the length LVH to specify the length of the vehicle.



2.3.4 Constraint Models (CONSTR)

Models associated with the model type (CONSTR) define constraints. For unconstrained optimization problems, no constraints are defined, and the CONSTR model type should not be specified.

2.3.4.1 Constraint Model 1 (CNSTRM1)

Description

CNSTRM1 is an optimization input model which defines one or more constraints. The data input to CNSTRM1 must conform to the optimization input format stated in Section 2.3.1. In addition, the following specifications must also be followed as a part of the CNSTRM1 input format. A required input for each constraint to be defined is a GTL expression statement. Each table listed as a part of the input data specified for CNSTRM1 must contain exactly one GTL expression statement. Also, an expression statement may optionally be a part of the CNSTRM1 input data.

If an expression statement is input with CNSTRM1, then this expression statement, along with any modifying slash statements or data assignment statements input with CNSTRM1, is assumed to define a single constraint. All quantities not specified assume their preset values. Furthermore, each table specified as a part of the CNSTRM1 input is assumed to contain data to define a single constraint. All quantities not specified as a part of the tabular input assume their preset values. That is, no data is carried over from one table to another nor is data input with CNSTRM1 carried over to the tables. As an illustration of this mechanism, consider Example 1. Four constraints are defined by this data specification--an altitude constraint, a latitude constraint, a longitude constraint, and a reentry angle constraint. The tolerance for the altitude constraint is 1. and the tolerance for the reentry angle constraint is 0.1. The tolerance for the other constraints is the preset value of CONTOL. Also, the input INBASIS = :YES: applies only to the reentry angle constraint.

If the data input with CNSTRM1 data does not contain a GTL expression,

then this data forms a set of data that is referred to as the model default data. Again, each table listed with the CNSTRM1 input must contain a single GTL expression and possibly may contain other data assignment statements or modifying slash statements. Each table defines a single constraint with any input quantity not specified as a part of the tabular input assuming the value specified in the model default data. If an input quantity is not specified either by tabular data or by model default data, then the preset value is assumed. Hence, if a GTL expression statement is not specified as a part of the data input with CNSTRM1, then such input data become the default values for the tables listed as a part of the CNSTRM1 input data. The individual tables are independent in that data listed within a table applies only to the constraint defined by that table. As an illustration of this mechanism, consider Example 2. This data specification defines three burn rate constraints. Since the data input with CNSTRM1 does not contain GTL expression statements, the specified slash statement and data assignment statements apply to the constraints defined by tables BRATE1, BRATE2, and BRATE3. Hence, the value of the quantity CONTOL is 0.0005 for the second and third constraints, while the input value of CONTOL = 0.3 is specified for the first constraint. Also, note that the input INBASIS = :YES: applies only to the second constraint.

All optimization and search operators assume that all constraints are of the form

$$c \geq 0 \qquad \text{or} \qquad c = 0$$

This form, however, is not particularly useful from a user's viewpoint. CNSTRM1 permits the constraints to be specified in any of the following forms:

$$q_1 = \text{constant}, \qquad q_1 \geq \text{constant}, \qquad q_1 \leq \text{constant}$$

$$q_1 = q_2, \qquad q_1 \geq q_2, \qquad q_1 \leq q_2$$

$$q_1 = q_2 \pm \text{constant}, \qquad q_1 \geq q_2 \pm \text{constant}, \qquad \text{or}$$

$$q_1 \leq q_2 \pm \text{constant},$$

where q_1 and q_2 are any GTS system variables. The conversion from all of these forms to the form $c \geq 0$ or $c = 0$ is performed internally in the computer program.

The algorithms encompassed in certain optimization and search operators (e.g., UOPTIM and UBEST) require partial derivative information. In this context the term partial derivative refers to the partial derivatives of the constraints with respect to the independent variables of the current problem. For reasons of accuracy and efficiency, if a method for analytical computation of the partial derivatives is available, then this method is advised. Otherwise, the partial derivatives are computed numerically. If the partial derivatives are not computed by the normal execution of the function generator, then the program permits a specialized routine to be executed for the computation of the partial derivatives. This routine is independent of an auxiliary routine that may be executed to evaluate the constraint. If numerical partials are to be computed, then the method (i.e., one side or two sided perturbation) and the initial perturbation size are specified as a part of the independent variable input. Obviously, the input data concerning the generation of the partial derivatives must be consistent between the constraints and independent variables.

Model Inputs

Each constraint is defined by an expression statement or a modified expression statement. The format for the expression is the following:

$$(q_1 \left[\begin{matrix}\text{.OF.}\\ \text{.AT.}\end{matrix}\right] \text{ qualifier 1} \left]\begin{matrix}\geq\\ =\end{matrix}\right| \; q_2 \left[\begin{matrix}\text{.OF.}\\ \text{.AT.}\end{matrix}\right] \text{ qualifier 2} \left] + \text{constant } [\text{units}]\right).$$

q_1 and q_2 are any GTS system variables. q_1 is a required input, and either q_2 or a constant is also a required input. The qualifiers are required only if q_1 and q_2 are not uniquely defined. The input "units" is an optional input required only to insure that q_1, q_2, and/or constant term all are expressed in the same units. The assumption is made that if a constant term is part of the definition of a constraint, then the units of the constant term are

compatible with the remaining terms of the constraint or that the constant term has the units specified by the input parameter "units".

The following modifying slash statements may be specified to further define the constraints. As with all modifying slash statements, these slash statements must immediately follow the expression statement. If a program variable is subscripted or qualified in the expression statement, then any reference to that program variable in a modifying slash statement must also include the subscript and/or qualifier. The following slash statements may reference either GTS system variable q_1 or q_2 or both, although only references to q_1 are shown.

The first slash statement is required only if q_1 is not computed by the normal execution of the function generator.

/ q_1 [[.OF. / .AT.] qualifier 1] IS COMPUTED BY routine name [data] /

The next two modifying slash statements specify the method of computation of the partial derivatives. The default method is numerical partials.

/ q_1 [[.OF. / .AT.] qualifier 1] PARTIALS ARE COMPUTED

ANALYTICALLY BY routine name [data] /

/ q_1 [[.OF. / .AT.] qualifier 1] PARTIALS ARE COMPUTED

NUMERICALLY /

The following independent slash statements may be input to CNSTRM1 (but not in a table referenced by CNSTRM1) as model default values, and as such will apply to all constraints subsequently referenced by CNSTRM1. Thus, these independent slash statements eliminate the need to specify modifying slash statements for each constraint.

/ALL FUNCTIONS ARE COMPUTED BY routine name [data] /

/ALL PARTIALS ARE COMPUTED ANALYTICALLY BY routine name [data] /

/ALL PARTIALS ARE COMPUTED NUMERICALLY/

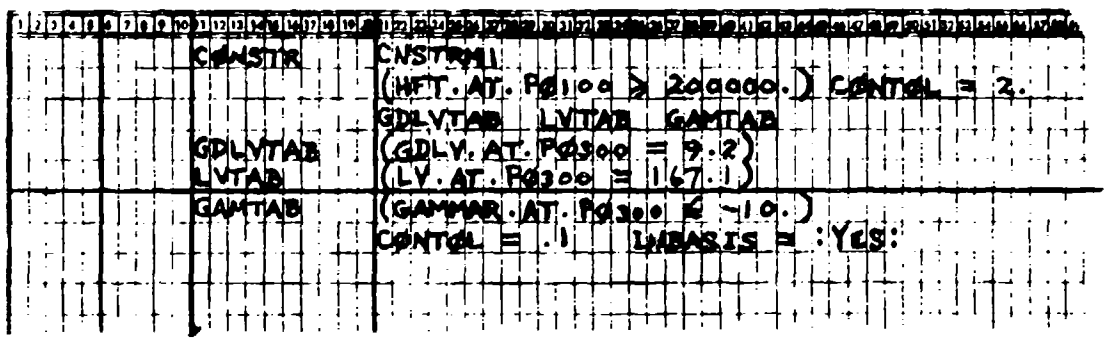
The following quantities are input via data assignment statements. The quantities need to be input only if the default values are not desired.

Mnemonic	Description	Default Value
CONTOL	Constraint tolerance A constraint is considered to be satisfied as an equality constraint if $\|c\| <$ CONTOL	0.001
INBASIS	Basis indicator INBASIS = :YES: The constraint is placed in the initial basis. INBASIS = :NO: This constraint is not in the initial basis.	:NO:
CONLIN	Nonlinearity factor of the constraints. If information is available about the linearity of the constraints, such as the constraints are linear or the constraints are highly nonlinear, then this information can be incorporated into the algorithm. $0. \leq \text{CONLIN} \leq 4.$ CONLIN = 0 implies constraints are linear. CONLIN = 4 implies constraints are highly nonlinear. (Currently only applicable to the UOPTIM operator.)	0.1

Mnemonic	Description	Default Value
CONSKL	Constraint scaling flag CONSKL = :SCALED: implies the constraints are scaled and the scale weights are computed internally. CONSKL = :UNSCALED: implies the constraints are not scaled. CONSKL = n.n implies the constraints are scaled and the scale weight for this constraint is n.n.	:SCALED:

Examples of CNSTRM1 input

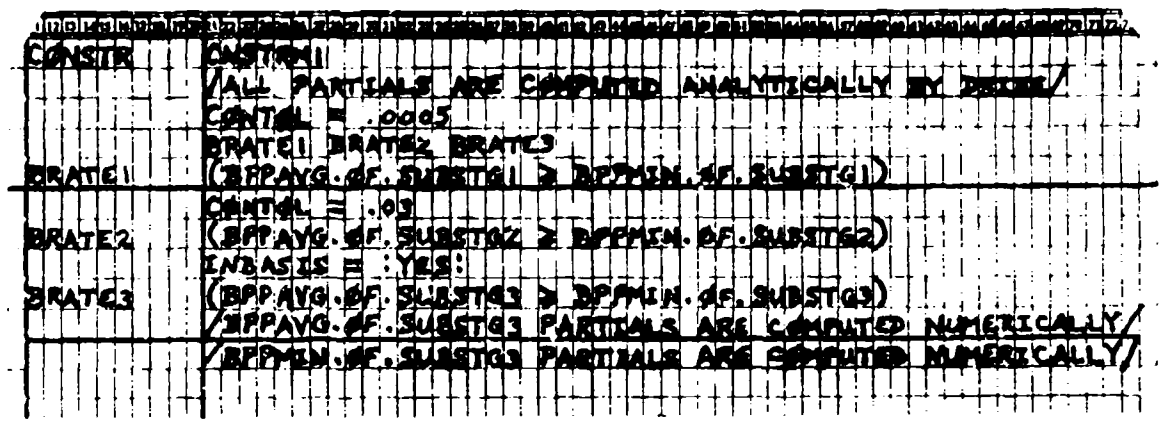
Example 1. Consider a trajectory optimization problem with four constraints. The impact point is constrained to satisfy a specified latitude and longitude, the altitude at event PC100 is required to be greater than 200,000 ft, and the reentry angle at event PC300 is required to be less than -10 deg. Also, assume the tolerance for the altitude constraint and the reentry angle constraint are different than the preset value, and assume the reentry angle constraint is to be in the initial basis.



Example 2. Consider a vehicle sizing problem with a burn rate constraint for each stage of a three stage vehicle. Further, assume that (1) the partial derivatives of the constraints for the first and second stage are computed

analytically by the routine DSIZE, but the third stage constraint requires numerical derivatives; (2) the tolerance for the first stage constraint is 0.03, but the tolerance for the other two stages is 0.0005; (3) the second stage constraint is to be placed in the initial basis.

Assume that data blocks named SUBSTG1, SUBSTG2, and SUBSTG3 contain the data corresponding to the three stages.



2.3.5 Independent Variable Model (INDVAR)

Models associated with the model type INDVAR define independent variables. An independent variable model is required for all optimization problems.

2.3.5.1 Independent Variable Model 1 (VARM1)

Description

VARM1 is an optimization input model which defines one or more independent variables. The data input to VARM1 must conform to the optimization input format stated in Section 2.3.1. In addition, the following specifications must be followed as a part of the VARM1 input format. A required input for each independent variable is a data assignment statement of the form VAR = [name of variable]. Each table listed as a part of the input data specified for VARM1 must contain a data assignment statement

of the form VAR = [name of variable]. Also, a data assignment statement of this form may be part of the input data specified for VARM1.

If a data assignment statement of this form is input as a part of the input data specified for VARM1, then this input along with any other data assignment statements input with VARM1, is assumed to be data which defines a single independent variable. All quantities not specified assume their preset values. Furthermore, each table specified as a part of the VARM1 input is assumed to contain data to define a single independent variable. All quantities not specified as a part of the tabular input assume their preset values. That is, data is not carried over from one table to another and data input with VARM1 is not carried over to the tables. As an illustration of this mechanism, consider Example 1. Three independent variables are defined by this data specification. The three independent variables are a launch azimuth (AZ1), a kick angle (KICK), and payload (PAYLOAD). All data specified via data assignment statements in the VARM1 input apply only to the independent variable AZ1, while the data specified in the tables VKICK and VPAYLD apply only to the independent variables KICK and PAYLOAD, respectively. That is, data specified for one independent variable is independent of the other variables.

Alternately, if the data input to VARM1 does not contain a data assignment statement of the form VAR = [name of variable], then all data assignment statements specified as a part of the input to VARM1 forms a set of data that is referred to as the model default data. Again, each table referenced as a part of the VARM1 input must contain a data assignment statement of the form VAR = [name of variable]. Each table contains data to define an independent variable with any input quantity not specified within the table assuming the value specified by the model default data. If an input is not specified by either the tabular input or the model default values, then the preset value is assumed. Hence, if a data assignment of the form VAR = [name of variable] is not specified as a part of the data listed with VARM1, then all such data assignment statements apply to the independent

variables defined in the tables listed as a part of the VARM1 input data. However, the individual tables are independent in that all data specified within a table applies only to the independent variable specified by that table. As an illustration of this mechanism, consider example 2. In this example, four independent variables are defined. Except for the upper and lower bounds (UPRBND and LWRBND) of the fourth independent variable (ALF4), all the input quantities which define the variables are identical. Hence, these quantities are specified as model input data which then apply to all the independent variables defined in the tables listed. Note, these model default values can be overridden by specifying the desired values as a part of the tabular data.

For the current implementation, all independent variable names must be GTL user-defined symbols. That is, the independent variable names must be defined by a GTL DEFINE statement or by the equivalence option. This mechanism implies that the independent variables are uniquely defined when referred to by the user-defined symbols. Thus, unlike the objective function or constraints, the independent variables do not require a qualifier to be uniquely defined. For trajectory optimization problems in which the partial trajectory option is specified, the computation time will be decreased if the event or phase where an independent variable becomes active is specified. This specification permits only that portion of the trajectory in which an independent variable is active to be simulated.

As a part of the independent variable input, the method by which the partial derivatives are computed is specified. In this context, the partial derivatives refer to the partial derivatives of the objective function and constraints with respect to the independent variables. This input information must be consistent with the input specified for the constraints and objective function. Thus, for example, if analytic computation of partial derivatives is specified by the independent variable input, either the required partial derivatives must be computed by the normal execution of the function generator or the routine which computes the partial derivatives must be specified as a part of the constraint and objective function input.

For the computation of partial derivatives via numerical perturbations, the current implementation is that the perturbation size for two-sided partials is determined at each point by the algorithm. For one-sided perturbations, however, the perturbation size remains constant at the input value.

Additional required inputs for each independent variable are upper and lower bounds. These input quantities define a region of computability for the optimization operator in that the iteration process will not exceed these bounds. These quantities should not be confused with constraints. In particular, the solution cannot be on a bound. Also, the bounds should not be made restrictive in an attempt to aid the optimization algorithm in finding a solution.

Model Inputs

Mnemonic	Description	Default Value
VAR	Name of the independent variable. This quantity must be input for each independent variable. No default value is allowed.	Must be input
STARTAT	Event at which variable is activated. Only applicable if the partial trajectories option has been specified for trajectory optimization problems.	Initial event
LWRBND	Lower bound on the independent variable.	Must be input
UPRBND	Upper bound on the independent variable.	Must be input
PDTYPE	Partial derivative type. PDTYPE = :1 SIDED: - one-sided perturbations are used to compute numerical partials.	:2SIDED:

Mnemonic	Description	Default Value
	PDTYPE = : 2 SIDED : - two-sided perturbations are used to compute numerical partials. PDTYPE = : ANALYTIC : - the partials are computed analytically.	
DELVAR	Initial perturbation size. Only applicable if one-sided or two-sided partials are requested.	(UPRBND-LWRBND)/1000.

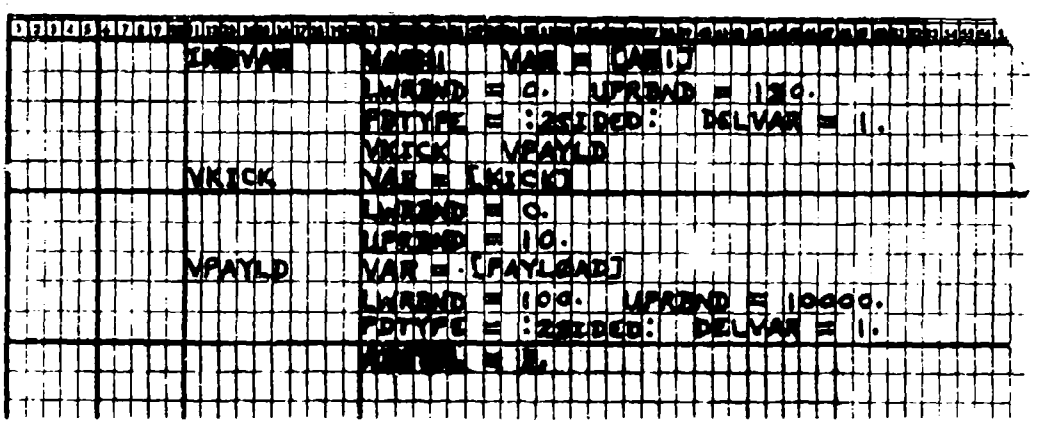
For the following quantities, the default values are recommended unless the user has an understanding of the program and a requirement to specify alternate input values.

Mnemonic	Description	Default Value
EPSBND	Epsilon bound for the independent variables The upper and lower bounds may be modified by this input. The upper and lower bounds used by the program are UPRBND + EPSBND and LWRBND - EPSBND. Thus EPSBND >0 expands the region of search while EPSBND < 0 contracts it.	0.001
SCALE	Scale flag. SCALE = : SCALED : - The independent variables are scaled. SCALE = : UNSCALED : - The independent variables are not scaled.	:SCALED:
RELTOL	Relative tolerance on the independent variables.	0.001

Mnemonic	Description	Default Value
ABSTOL	Absolute tolerance on the independent variable.	0.001
GRADTOL	Gradient tolerance.	0.1

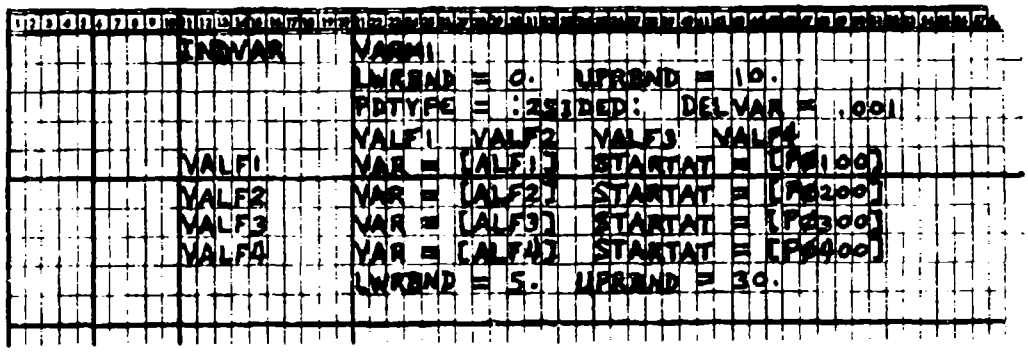
Examples of VARM1 Input

Example 1. Consider a trajectory optimization problem with three independent variables. The independent variables are the initial azimuth, the kick angle, and the payload. Assume that two-sided partial derivatives are requested for all the variables, but the default value for the initial perturbation size is overridden for the azimuth and payload, and assume an absolute tolerance of 1. is desired for the payload. Note that the names of the independent variables are user-defined names and not program variable names, and note that the upper and lower bound is specified as required, for each independent variable.



Example 2. Consider a trajectory optimization problem with four independent variables. The independent variables are the value of the angle of attack at four points along the trajectory. Assume that the partial trajectory option is specified and the four values of angle of attack are specified at PO100, PO200,

PO300, and PO400. Also, assume the upper and lower bounds are the same for the first three variables, and assume that two-sided derivatives with an initial perturbation size of 0.001 are desired for all the variables.



2.4 Optimization and Search Output

Selected information is printed during the optimization process. Since the purpose of the output is to permit the user to analyze the optimization process, a portion of the information printed is specific to the individual optimization or search operator being executed. This information is discussed in Section 2.2 as a part of the description of the individual operators. In addition, information is printed which is identical for all operators. This information includes a summary of the input data which defines the optimization or search problem, and a summary of the information obtained at each point where an evaluation of the objective function, constraints, and gradients is made. The following section describes only the output which is common to all operators.

2.4.1 Optimization Input Summary

Initially, before the iteration process is started, a summary of the user-defined input data is printed. All data output at this point should be the same as input. Scaling or other internal processing is not reflected in this print. The summary print is divided into three parts. The first block contains a description of the objective function. The format for this information is the following:

OBJECTIVE FUNCTION

$$\begin{Bmatrix}\text{MAXIMIZE}\\ \text{MINIMIZE}\end{Bmatrix} \text{ name of objective function } \begin{bmatrix}\text{.AT.}\\ \text{.OF.}\end{bmatrix} \text{ qualifier}$$

The name of the objective function and the qualifier are the same names as the user has specified. If a search or root finding problem is being defined, then one of the following messages is printed, depending on the type of problem the user has specified.

SEARCH PROBLEM - NO OBJECTIVE FUNCTION

OBJECTIVE FUNCTION IS THE SUM OF THE SQUARES OF THE INDEPENDENT VARIABLES

The second block of information contains a summary of the input data defining the constraints in the following format:

m CONSTRAINTS

1. (q_1 [.OF. / .AT.] qualifier 1 {.GE. / .EQ. / .LE.} q_2 [.OF. / .AT.] qualifier 2 ± constant) tolerance

.
.
.

m. (q_1 [.OF. / .AT.] qualifier 1 {.GE. / .EQ. / .LE.} q_2 [.OF. / .AT.] qualifier 2 ± constant) tolerance

The final block contains a summary of the independent variable input in the following format:

n VARIABLES

VARIABLE - INITIAL VALUE	LOWER BOUND	UPPER BOUND	DERIVATIVE TYPE
1. name of variable IN event name $x_o(1)$	lb(1)	ub(1)	{ONE SIDED / TWO SIDED / ANALYTIC}
⋮			
n. name of variable IN event name $x_o(n)$	lb(n)	ub(n)	{ONE SIDED / TWO SIDED / ANALYTIC}

VARIABLE	ABSOLUTE ERROR	RELATIVE ERROR	GRADIENT TEST	EPISILON BOUND
1.	ae(1)	re(1)	grad. (1)	eb(1)
⋮	⋮	⋮	⋮	⋮
n.	ae(n)	re(n)	grad. (n)	eb(n)

2.4.2 Iteration Summary

At each point where an evaluation of the objective function constraints

and gradients is requested, a summary of information computed at that point is printed.

The format for this information is the following:

OBJECTIVE FUNCTION

(name of objective function) = xx.x n_1 PERTURBATIONS n_2 FUNCTION EVALUATIONS

CONSTRAINTS

c(1) = value of c(1)... (name of q_1 = xx.x) {.GE. .EQ. .LE.}

(name of q_2 = xx.x) ± constant term

.
.
.

c(m) = value of c(m)... (name of q_1 = xx.x) {.GE. .EQ. .LE.}

(name of q_2 = xx.x) ± constant term.

VARIABLES

VARIABLE NAME	VALUE	PERTURBATION SIZE
1. name of variable 1	value of variable 1	value of perturbation size for variable 1
	.	
	.	
	.	
n. name of variable n	value of variable n	value of perturbation size for variable n

n_1 is a cumulative total of the number of perturbed evaluations that have been made. n_2 is a cumulative total of the number of points at which an evaluation of objective function, constraint, or gradient information has been requested. No information is printed for perturbed evaluations nor is

any gradient information printed.

The value of c(I) is the value of the constraint in the form required by the optimization operator (i.e., $c(I) \geq 0$ or $c(I) = 0$). Thus, for example, if the I-th constraint is

(H. AT. PO300 . LE. HA. AT. PO200 + 10.)

and if at the current point H (at event PO300) = 100 and HA (at event PO200) = 80, the corresponding output at this point would be

c(I) = -10 ... (H = 100) . LE. (HA = 80) + 10.

In addition, an asterisk (*) will appear at the left of all constraints that are in the current basis. Qualifiers for the objective function or the constraints are not printed in the point-by-point output.

SECTION 3

INTEGRATION OPERATORS

Consider the mathematical problem of solving a system of ordinary differential equations with given initial conditions

$$\frac{dy(t)}{dt} = f(t, y) \qquad y(t_o) = y_o. \tag{1}$$

where y and f are vector valued functions. If $f(t, y)$ is assumed to be continuous with respect to t and y, then several numerical techniques are available to compute a solution to the system (1). These numerical techniques compute a solution to (1) in the sense that they provide a scheme for obtaining a numerical approximation to the value of $y(t)$ for $t \neq t_0$. The GTS system contains several methods for solving differential equations. These techniques can be applied to any dynamic system; that is, any system that is represented by a set of differential equations of the type (1).

The primary application of the integration techniques, however, is for providing a trajectory simulation capability, and the integration techniques incorporated in the GTS program were chosen for their viability for performing trajectory simulations. Specifically, the GTS program provides the following techniques:

(i) 4-th order fixed step Runge-Kutta
(ii) m-th order (m = 1 to 8) fixed step Adams-Moulton
(iii) m-th order (m = 1 to 8) variable step Adams-Moulton

A brief description of these methods is contained in Sections 3.3.1, 3.3.2, and 3.3.3. A more complete mathematical description can be found in any standard text on numerical analysis, such as Ref. 5 or Ref. 6. Clearly, any one of these methods is not suitable for all applications. Each method has advantages and disadvantages. Consequently, Section 3.4 contains recommendations concerning the types of problems for which each method may be best suited.

The input data required to specify a particular method is input via an integration model corresponding to the model type INTGRA. Currently,

a single model, INTGRM1, encompasses all the available methods. This model is discussed in Section 3. 4. 1.

3.1 Trajectory Simulations

A major requirement of a trajectory simulation capability is the ability to solve differential equations of the type (1). Several considerations must be recognized with respect to the integration process. The evaluation of the derivatives, that is, the evaluation of the function $f(t, y)$ is accomplished by the orderly execution of the user-specified engineering models (e.g., propulsion, control, equations of motion). It is important to realize that it is the model configuration which the user has specified that characterizes a trajectory and not the integration technique that is applied.

Besides providing freedom for the user to select the models he desires, the independence of the integration operator and engineering models also permits new engineering models to be added to the program model library and permits modifications to be made to the integration operators without requiring changes to the existing engineering models. By the specification of models, the user also determines the number of equations to be integrated. In addition to three degree-of-freedom or six degree-of-freedom simulations, the user may request that auxiliary equations of interest, such as ideal velocity or velocity losses, be integrated. Again, the specification of the number of equations to be integrated is part of the model selection process and independent of the integration operator.

In addition to the final result, discrete time points along the trajectory, or events, may also be of interest. As is documented in Volume II, GTS provides a flexible method for event specification. If such an event has been detected, the response to this event may be to alter the model configuration or to introduce new data to the existing models. Mathematically, these actions may introduce discontinuities in the function $f(t, y)$ of Eq. (1). Hence, the integration process cannot proceed directly. Rather, the integration operator must essentially restart the integration process at that point in order for the integration operator to perform correctly.

3.2 Integration Methods

Assume, for definiteness, a first order differential equation with specified initial conditions of the form

$$\frac{dy}{dt} = f(t, y) \qquad y(t_o) = y_o$$

and assume the value of y is desired at some time $t_f \neq t_o$. Note, t_f may be less than t_o. Briefly, the method by which a numerical approximation to the value of $y(t_f)$ is obtained is to make a functional approximation to the function y. Clearly, a single approximation is not likely to be valid for the entire interval t_o to t_f. Consequently, we consider an approximation over a smaller interval t_o to $t_o + \Delta t$. The size of the increment Δt may be user-specified or may be computed within the algorithm.

The basic integration method then proceeds as follows. First, $y(t_o)$ and $\dot{y}(t_o) = f(t_o, y_o)$ are known. Then obtain an approximation y_1 to the value of y at $t_1 = t_o + \Delta t$. $\dot{y}_1 = f(t_1, y_1)$ can then be evaluated. The values y_1 and $\dot{y}_1$ provide initial values to begin the process again in order to obtain an approximation to y at $t_2 = t_1 + \Delta t$. This process continues until $t_N = t_f$, for some N. Given values y_n and $\dot{y}_n$, the method by which an approximation y_{n+1} to the value $y(t_{n+1})$ is generated, distinguishes the various integration schemes. A brief outline of the methods available in GTS are given in the following sections.

3.2.1 Runge-Kutta, Fixed Step (4-th Order)

Let y_n and $\dot{y}_n$ be approximations to $y(t_n)$ and $\dot{y}(t_n)$. y_n is available either from the initial conditions or from the previous integration step. $\dot{y}_n$ can be computed from y_n by $\dot{y}_n = f(t_n, y_n)$. The function of the integration technique is to obtain approximations y_{n+1} and $\dot{y}_{n+1}$ to the values of $y(t_{n+1})$ and $\dot{y}(t_{n+1})$, where $t_{n+1} = t_n + \Delta t$. One method to obtain such an approximation is to expand $y(t)$ in a Taylor's series. This technique would require the derivatives $\dot{y}(t_n)$, $\ddot{y}(t_n)$, $\dddot{y}(t_n)$ ... but this computation may be quite cumbersome. The Runge-Kutta method evaluates

the first five terms of the Taylor's series expansion, but without the necessity of higher order derivative evaluations. This approximation is obtained by the evaluation of $\dot{y} = f$ at selected points in the interval. The mathematical details are eliminated; however, the 4-th order Runge-Kutta recursion formula is

$$y_{n+1} = y_n + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$k_1 = \Delta t \cdot f(t_n, y_n)$$

$$k_2 = \Delta t \cdot f(t_n + \frac{\Delta t}{2}, y_n + \frac{1}{2} k_1)$$

$$k_3 = \Delta t \cdot f(t_n + \frac{\Delta t}{2}, y_n + \frac{1}{2} k_2)$$

$$k_4 = \Delta t \cdot f(t_n + \Delta t, y_n + k_3)$$

This method is illustrated schematically in Figure 3.2-1.

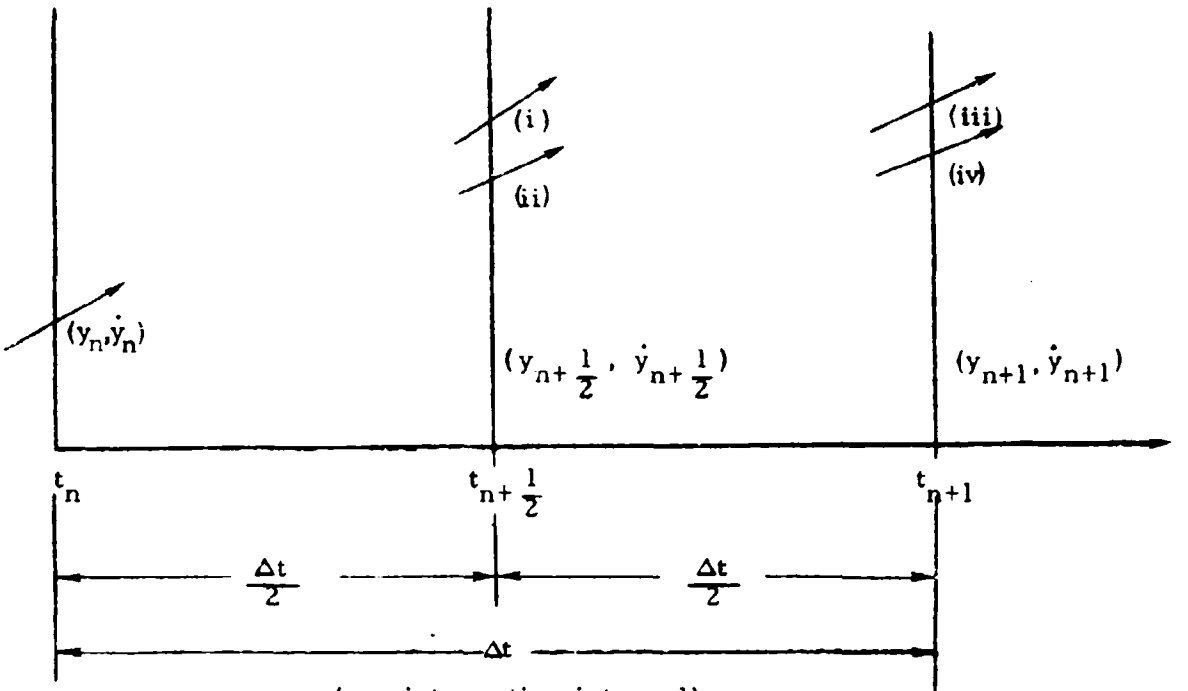
To complete one cycle of this integration scheme (i.e., to obtain y_{n+1} and $\dot{y}_{n+1}$), it is necessary to evaluate the function f(t, y) four times. For applications such as trajectory simulations in which this evaluation is time consuming, the overall efficiency may be reduced when compared to the Adams-Moulton method (see Section 3.3.2).

An important advantage of the Runge-Kutta method is that it is self-starting. That is, it only requires values of y and $\dot{y}$ at the point t_n to obtain values of y_{n+1} and $\dot{y}_{n+1}$. This characteristic implies that the Runge-Kutta process can begin at any point, such as an event, without requiring values from previous integration intervals.

3.2.2 Adams-Moulton, Fixed Step (m-th Order)

The Adams-Moulton method is one of the class of methods referred to as predictor-corrector methods. Whereas the Runge-Kutta method requires

(cont. p. 3.2-4)



(one integration interval)

i) y_n and $\dot{y}_n$ are given, $y_{n+\frac{1}{2}}$ is an estimate of $y(t_n + \frac{\Delta t}{2})$ which is computed as a function of y_n and $\dot{y}_n$ by $y_{n+\frac{1}{2}} = y_n + \frac{\Delta t}{2} \dot{y}_n$. The derivative $\dot{y}(t_n + \frac{\Delta t}{2})$ is then estimated by $\dot{y}_{n+\frac{1}{2}} = f(t_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})$.

ii) $\bar{y}_{n+\frac{1}{2}}$ is a refined estimate of $y(t_n + \frac{\Delta t}{2})$ which is computed by $\bar{y}_{n+\frac{1}{2}} = y_n + \frac{\Delta t}{2} \dot{y}_{n+\frac{1}{2}}$, and a refined estimate of $\dot{y}(t_{n+\frac{1}{2}})$ is computed by $\bar{\dot{y}}_{n+\frac{1}{2}} = f(t_{n+\frac{1}{2}}, \bar{y}_{n+\frac{1}{2}})$.

iii) y_{n+1} is an estimate of $y(t_{n+1})$ which is computed by $y_{n+1} = y_n + \Delta t \, \bar{\dot{y}}_{n+\frac{1}{2}}$, and the derivative $\dot{y}(t_{n+1})$ is estimated by $\dot{y}_{n+1} = f(t_{n+1}, y_{n+1})$.

iv) $\bar{y}_{n+1}$ is a final estimate of $y(t_{n+1})$ which is computed by $\bar{y}_{n+1} = y_n + \frac{\Delta t}{6} (\dot{y}_n + 2\dot{y}_{n+\frac{1}{2}} + 2\bar{\dot{y}}_{n+\frac{1}{2}} + \dot{y}_{n+1})$.

Figure 3.2-1 The 4-th Order Runge-Kutta Integration Method

information only at one point, t_n, in order to proceed to the next point, t_{n+1}, the predictor-corrector schemes use information at more than one point to obtain the approximation at t_{n+1}.

Assume that the approximate values of y and $\dot{y}$ are known at the set of m points t_i, $i = n, n-1, \ldots, n-m+1$. The Adams-Moulton predictor-corrector scheme proceeds by determining the polynomial which interpolates the values of $f(t, y)$ at the points t_i, $i = n, n-1, \ldots, n-m+1$. This polynomial is extrapolated to time t_{n+1} and an estimate of $y(t_{n+1})$ is obtained by integrating this extrapolating polynomial. This is the predictor step. The corrector step then refines the estimate of $y(t_{n+1})$ by the iterative solution of an implicit function involving $\dot{y}_{n+1}$.

When the time points are equally spaced, the mathematical formulas for the m-th order Adams-Moulton scheme are

(i) Predictor formula (Adams-Bashforth)

$$y_{n+1}^{(1)} = y_n + \Delta t \left(a_{m,0} \dot{y}_n + a_{m,1} \dot{y}_{n-1} + \ldots + a_{m,m-1} \dot{y}_{n-m+1}\right) \quad (1)$$

(ii) Corrector formula (Adams-Moulton)

$$y_{n+1} = y_n + \Delta t \left(b_{m,0} \dot{y}_{n+1}^{(1)} + b_{m,1} \dot{y}_n + b_{m,2} \dot{y}_{n-1} + \ldots \right. \quad (2)$$

$$\left. + b_{m,m-1} \dot{y}_{n-m+2}\right)$$

where

$$\dot{y}_{n+1}^{(1)} = f(t_{n+1}, y_{n+1}^{(1)})$$

The coefficients are determined by

$$a_{m,q} = (-1)^q \sum_{k=q}^{m} \binom{k}{q} \gamma_k^{(a)} \quad (3)$$

where $\gamma_k^{(a)}$ is generated recursively by

$$\gamma_k^{(a)} + \frac{1}{2}\gamma_{k-1}^{(a)} + \frac{1}{3}\gamma_{k-2}^{(a)} + \ldots + \frac{1}{k+1}\gamma_0^{(a)} = 1 \qquad (4)$$

$$k = 0, 1, 2, \ldots$$

$$b_{m,q} = (-1)^q \sum_{k=q}^{m} \binom{k}{q} \gamma_k^{(b)} \qquad (5)$$

where $\gamma_k^{(b)}$ is generated recursively by

$$\gamma_k^{(b)} + \frac{1}{2}\gamma_{k-1}^{(b)} + \frac{1}{3}\gamma_{k-2}^{(b)} + \ldots + \frac{1}{k+1}\gamma_0^{(b)} = \begin{cases} 1, k = 0 \\ 0, k > 0 \end{cases} \qquad (6)$$

When the time points are not equally spaced, the interpolating polynomial is integrated to obtain $a_{m,i}$ and $b_{m,i}$, $i = 0, 1, \ldots, m-1$. The Adams-Moulton method is illustrated in Figure 3.2-2.

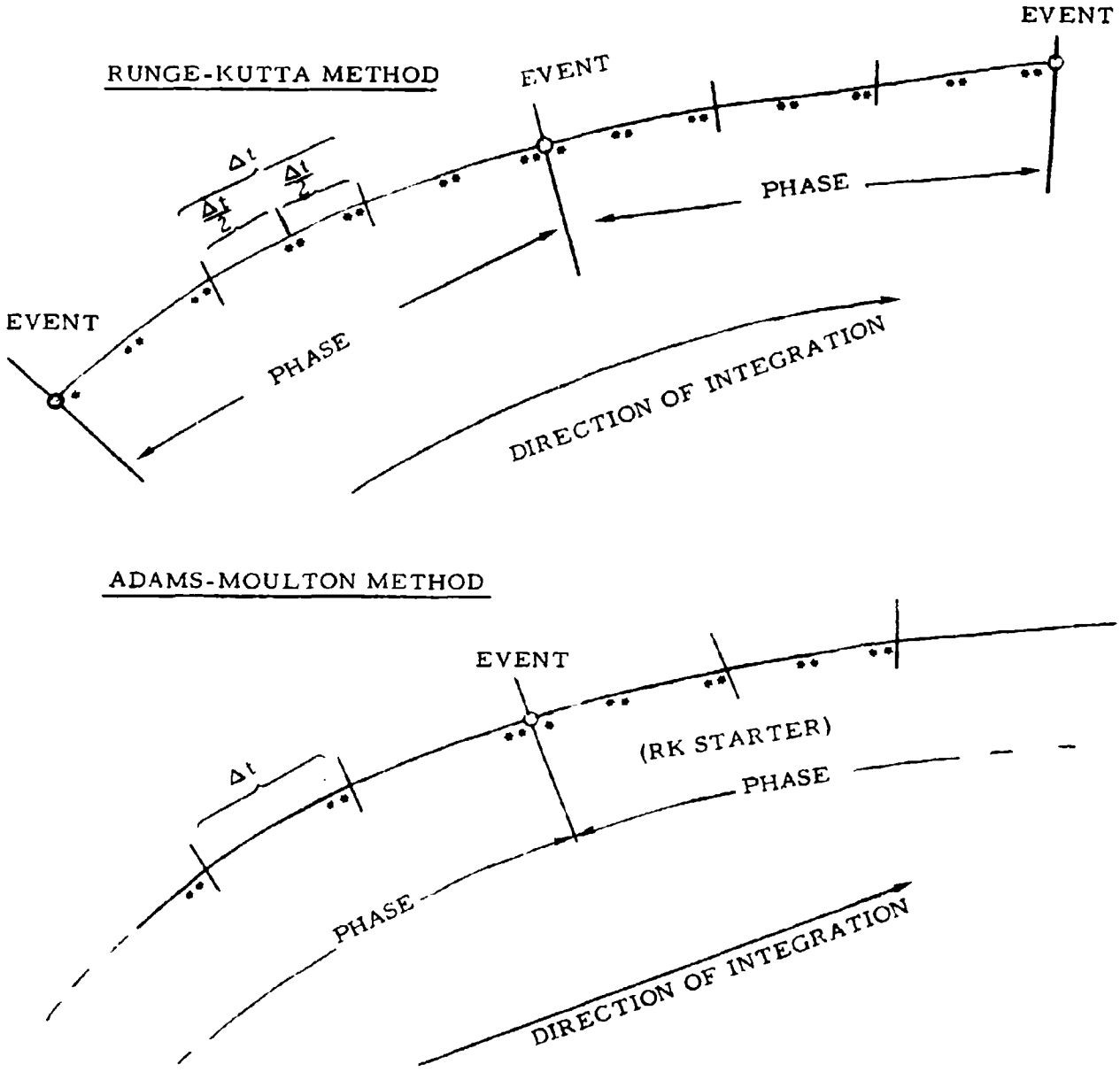
A major advantage of the Adams-Moulton method is that it requires only two evaluations of the function f (t, y) per integration step. Hence, for a given step size it should be more efficient than the Runge-Kutta scheme, which requires four such evaluations. However, the Adams-Moulton method does require the values of y_n and $\dot{y}_n$ at points outside the current integration intervals which must be generated by some other process. Within GTS, the Runge-Kutta method is used to provide the initial information required by the Adams-Moulton scheme.

The fact that the Adams-Moulton scheme requires information from previous integration intervals implies that this method is more sensitive to changes in the derivatives from one integration interval to the next. Also the dependency on past information implies that the Adams-Moulton method must be restarted at each point where there is a discontinuity in the derivatives (i.e., the function f (t, y)). For example, at the beginning of each phase, the Adams-Moulton method must be restarted. This situation is illustrated in Figure 3.2-3. Note that an extra evaluation of the derivatives is required at the right side of each event which introduces discontinuities into the derivatives.

i) Initially, y_n, $\dot{y}_n$, $\dot{y}_{n-1} \cdots \dot{y}_{n-(m-1)}$ are available. $y_{n+1}^{(1)}$ is an estimate of $y(t_{n+1})$ which is computed by a "predictor" formula using y_n, $\dot{y}_n$, $\dot{y}_{n-1} \cdots \dot{y}_{n-(m-1)}$ (see Eq. (1)). An estimate of the derivative $\dot{y}_{n+1}^{(1)} = f(t_n, y_{n+1}^{(1)})$ is then evaluated.

ii) y_{n+1} is a refined estimate of $y(t_{n+1})$ which is computed by a "corrector" formula using y_n, $\dot{y}_{n+1}^{(1)}$, $\dot{y}_n$, $\dot{y}_{n-1} \cdots \dot{y}_{n-(m-2)}$ (see Eq. (2)). The derivative estimate $\dot{y}_{n+1}$ is then reevaluated.

Figure 3.2-2 Adams-Moulton Integration Metnod



*DERIVATIVE EVALUATION

Figure 3.2-3. Summary of Derivative Evaluation

3.2.3 Adams-Moulton, Variable Step (m-th Order)

An advantage of the Adams-Moulton method, as contrasted with the Runge-Kutta method, is that an error estimate is easily obtained. This error estimate can then be examined to determine whether the current integration step size is adequate for desired accuracy. Specifically, using the notation of Eqs. (1) and (2), the error estimate made within GTS for the k-th integration step is

$$E_{n+1} = c \frac{|y_{n+1}^{(1)} - y_{n+1}|}{\min(|y_{n+1}|, 1)} \tag{7}$$

where c is a computed constant.

Assume that a maximum error E_{max} and a minimum error E_{min} have been established, and assume E_{n+1} has been computed by Eq. (7). Then the integration step size Δt is determined as follows:

i) If $E_{min} \leq E_{n+1} \leq E_{max}$, proceed to the next integration step.

ii) If $E_{n+1} > E_{max}$, the step size is too large to maintain the desired accuracy, the step size is halved, and another integration step is taken from t_n.

iii) If $E_{n+1} < E_{min}$, then the step size is so small that more accuracy is being maintained than desired, and the step size is doubled.

Note, however, if the step size is changed, the coefficients for the Adams-Moulton method (see Eqs. (3), (4), (5), (6)) must be recomputed. Within GTS, E_{max} may be input, but E_{min} is computed by

$$E_{min} = \frac{E_{max}}{2^{m+4}} \tag{8}$$

Also, for the GTS implementation, the step size is not permitted to exceed specified upper and lower limits. That is, if $E_{n+1} < E_{min}$ but the integration step size is equal to the maximum permitted step size Δt_{max}, then the integration step size is not increased but is maintained at the current

value. Conversely, if $E_{n+1} > E_{max}$ but the integration step size is equal to the minimum permitted value Δt_{min}, then the integration step size is not reduced. Rather, the job is terminated since the error criterion cannot be satisfied and the step size is the minimum permitted.

3.3 Recommendations for Usage

The proper choice of an integration method to be used depends on a number of complex and interrelated factors. Some of these factors are:

1. The type of overall problem; e.g., as indicated in Section 2.1, trajectory optimization problems may require more accurate integration than would be required if an optimization operator were not being executed.

2. The type of dynamics problem being simulated as defined by the functions (model types) selected to be simulated; e.g., a six-degree-of-freedom simulation of an RV introduces more complex dynamics than would a point mass simulation of the same RV.

3. The model selected to simulate a specific function; e.g., the attitude model ATTM1 introduces six fast differential equations into the system, while model ATTM3 does not introduce any differential equations.

4. The continuity of the differential equations in the system; e.g., some models are formulated with an iteration loop through some of the equations in the model. This usually does not appear as a continuous system of equations to the integration operator.

5. The smoothness or continuity of the tabular data being used.

The smoothness, or lack of smoothness, in the tabular input data is particularly important. This is because a table point which falls within an integration step may introduce a discontinuity into the derivatives, and the integration techniques available in GTS cannot be used effectively across discontinuities in the derivatives.

For tables which contain explicit functions of time as the independent variable the GTS system automatically forces each table point to be an end point of an integration interval; consequently, a table point never falls within an integration step. See Section 4.3 for the details of this table timing mechanism. As discussed in Section 3.3.1, the Runge-Kutta method is self-starting, so the table timing mechanism assures that such tables will not introduce a numerical discontinuity within a given step; that is, derivatives will be continuous across a given step. While this is an extremely effective mechanism when using Runge-Kutta integration, its effectiveness is reduced when using a predictor-corrector method, since table points can occur in the set of steps used by the predictor-corrector method.

Experience has indicated that the following guidelines on numerical integration are appropriate for the simulation of trajectories.

A. Powered Flight Trajectories

(1) Vehicles with tabular thrust data:
Runge-Kutta integration is suggested, and a nominal integration interval of 1 sec is usually adequate.

(2) Vehicles with constant thrust data and in the lower part of the atmosphere; e.g., below 150,000 ft:
Runge-Kutta integration is suggested, and a nominal integration interval of 1 sec is usually adequate.

(3) Vehicles with constant thrust data and in the upper part of the atmosphere; e.g., above 150,000 ft:
A 4th, 5th, or 6th order variable step predictor-corrector method should be more effective than Runge-Kutta. An initial integration interval of 2 sec should be adequate.

B. Nonpowered Flight

(1) Point mass and three degree-of-freedom trajectories:
A variable step predictor-corrector method should be more efficient than Runge-Kutta. An 8-th order Adams-Moulton method is suggested. An initial integration interval between 0.5 and 4 sec is suggested. The integration operator will change the value of the integration step as required. The integration interval may vary from 256 sec or more for orbital trajectories, to 0.125 sec or less for RV's as they approach the surface of the earth. The initial integration interval should not be greater than 4 to 10 sec.

(2) Six degree-of-freedom trajectories:
A variable step predictor-corrector method should be more efficient than Runge-Kutta. A 4-th order method is suggested if the vehicle is spinning or tumbling and tabular aerodynamic data is used; otherwise, a higher order method should be more efficient. The initial integration interval may be obtained from the GTS model COMDTIN (COMpute DTIN). This model should be associated with the initialization model type (INIT), in the first appropriate phase data block. This model does not require any intra-model input.

As discussed in Section 3.3.3, for the variable step predictor-corrector method, the integration step size, in part, is controlled by E_{max}, which may be input (the mnemonic is ER). The preset value of ER is reasonable for most problems; however, it should be smaller for search or optimization problems which require gradient information, e.g., OPTIM or BEST. A value of $1. \times 10^{-9}$ should be adequate.

As stated earlier, the above suggestions for integration usage are guides and are not to be construed as absolute rules. New users of the GTS system, or users who are uncertain about the integration method for a particular application, are urged to contact or consult with a cognizant GTS programmer-analyst.

3.4 Integration Input Models (INTGRA)

The input data required to specify the integration operator is input via a model associated with the integration model type INTGRA.

For all problems which require an integration operator, e.g., trajectory simulations, the integration model type INTGRA and an associated model must be specified.

3.4.1 Integration Model 1 (INTGRM1)

Description

INTGRM1 is currently the only model available to specify the integration operator. All the methods discussed in Section 3.2 and all data associated with these methods are specified with this model.

Mnemonic	Description	Preset
METHOD	Integration method =:RK FIXED STEP: specifies fixed step Runge-Kutta integration =:AM FIXED STEP: specifies fixed step Adams-Moulton integration =:AM VARIABLE STEP: specifies variable step Adams-Moulton integration	:RK FIXED STEP:
ORDER	Order of the Adams-Moulton integration--either fixed or variable step. Applicable for Adams-Moulton integration only. (integer input)	4
DTIN	The input nominal Δt, in seconds, serves as the initial integration step size over a phase. DTIN must be input for the initial phase. If DTIN is not input for a subsequent phase, DTIN from the previous phase carries over.	None

Mnemonic	Description	Preset
DTMAX	Maximum allowable integration step size.	128.
DTMIN	Minimum allowable integration step size.	2^{-17}
ER	Upper bound for integration error associated with the AM variable step integration method. Applicable only for variable step AM.	5×10^{-6}

Examples of INTGRM1 input

```
PH10   4-TH ORDER RUNGE-KUTTA INTEGRATION - FIXED STEP
INTGRA   INTGRM1   METHOD = :RK FIXED STEP:
                   DTIN = 2.

PH20   5-TH ORDER ADAMS-MOULTON INTEGRATION - FIXED STEP
INTGRA   INTGRM1   METHOD = :AM FIXED STEP:
                   DTIN = 30.
                   ORDER = 11

PH30   4-TH ORDER ADAMS-MOULTON INTEGRATION - VARIABLE STEP
INTGRA   INTGRM1   METHOD = :AM VARIABLE STEP:
                   ORDER = 41
                   DTIN = .0625
                   DTMAX = 1.
                   DTMIN = .0001
                   ER = 5.E-5
```

3.5 Integration Output

Information concerning the integration operator is printed in two forms. The value of the integration step size (mnemonic DTNOM) is printed as part of the standard block print. Secondly, a summary of the error computations and integration step size is printed for the fixed step and variable step Adams-Moulton methods.

For the Adams-Moulton fixed step method, the following information is included in the integration summary print. The name and corresponding trajectory time of each event which occurred in the simulation is printed. Between the listing of the events, a message is printed if either the computed error for one integration step is greater than the maximum error (the input quantity ER) or is less than the minimum error (see Eq. (8)). Included in this message are the current trajectory time and the current step size. Also, if the error is too large, the integrated variable for which the computed error was the largest for the current integrated interval is printed. This message which indicates the step size was too large or too small is suppressed after four successive occurrences.

The format for this integration summary is the following:

```
                    •
                    •
                    •
EVENT   event name   TIME = xx.xxx          event title

        |RK START UP                |
        |RK START UP NOT REQUIRED|   DTNOM = xx.xx

        time         step size   FIXED STEP |ERROR>ERMAX...variable|
                                            |ERROR<ERMIN           |
                    •
                    •
                    •
        time         step size   FIXED STEP |ERROR>EPMAX...variable|
                                            |ERROR<ERMIN           |

EVENT   event name   TIME = xx.xxx          event title
                    •
                    •
                    •
```

For the Adams-Moulton variable step method, the following information is included in the integration summary print. The name and corresponding trajectory event time for each event which occurred in the simulation is printed. A message is printed every time the integration interval is halved or doubled.

Included in these messages are the trajectory time, the current integration interval, and an indication if a Runge-Kutta start up is required. Also, if the integration interval is halved, the integrated variable with the largest computed error for the current integration interval is printed.

Successive halving or doubling of the integration interval results in an integration interval which is equal to (2^i)DTIN, i=0, ±1, ±2, . . . , where DTIN is the initial integration interval. This integer i is printed for every integration step. The format for this integration summary is the following:

```
                    •
                    •
                    •
EVENT    event name     TIME = xx.xx        event title

  |RK STARTUP                                |
  |RK STARTUP NOT REQUIRED|          DTNOM = xx.xx

   time          step size          |HALVE . . . variable|
                                    |DOUBLE              |
                    •
                    •
                    •
time      i  i  i     •••                          i

   time          step size          |HALVE . . . variable|
                                    |DOUBLE              |
                    •
                    •
                    •
time      i  i  i     •••                          i

EVENT    event name     TIME = xx.xx        event title
                    •
                    •
                    •
```

SECTION 4

INTERPOLATION OPERATORS

The interpolation operators provide a capability for solving the following problem. Given a finite number of values of a function f(x) corresponding to $x_1, x_2 \ldots x_n$, determine the value of f(x) at points other than the given points. Currently, in GTS the primary application of the interpolation operators is for processing tabular data. As is documented in Volume II, the GTL tabular input format provides a convenient format for specifying the tabular data, and a description of the input options is not repeated here. Rather, this section contains a brief description of the method which generates values of the function f(x). A more complete mathematical description of the interpolation process can be found in most texts on numerical analysis, such as Ref. 7 and Ref. 8.

4.1 Interpolation Formulas

For discussing the interpolation scheme, assume that the values of the independent variables and dependent variables reflect any user requested scaling or biasing. The interpolation is performed by a Lagrange interpolation formula. The exact form of the formula depends on the number of independent variables and the order of interpolation.

4.1.1 Univariate Interpolation

Consider the formula for a single independent variable. Assume that the order of the interpolation is m. m-th order interpolation requires m+1 distinct values of the independent variable $x_0, x_1 \ldots x_m$ and the corresponding function values $f(x_i)$, $i = 0, \ldots, m$.

The basic interpolation procedure is to determine the polynomial p_m of degree less than or equal to m such that

$$p_m(x_i) = f(x_i) \qquad i = 0, 1, \ldots, m$$

The value of f at an arbitrary point x is approximated by evaluating $p_m(x)$.

If N represents the total number of points input in the table, then N must be greater than or equal to m+1. Otherwise, no interpolation is performed. Assume the value of f at an arbitrary point x is requested, then m+1 points, $x_0, x_1 \ldots x_m$, must be selected from the total of N points. These points are selected as follows. If the order of interpolation m is odd, then the (m+1)/2 tabular points immediately less than or equal to x and the (m+1)/2 tabular points immediately greater than x are selected. If the order of interpolation m is even, then m/2 tabular points immediately less than or equal to x and the (m+2)/2 immediately greater than x are selected for the interpolation. That is, the points $x_0, x_1 \ldots x_m$ are chosen such that

$$x_0 < x_1 < \ldots < x_k \leq x < x_{k+1} < \ldots < x_m$$

where $k = (m + \text{mod}(m, 2)/2)$

If x is less than the k-th point, where k is given above, of the original input set of points, then the first m+1 tabular points are chosen. Likewise, if x is greater than the $(N-k)^{th}$ point of the original input set of points, then the last m+1 points of the table are chosen.

Given this selected set of tabular points, $x_0, x_1, x_2 \ldots x_m$, along with the corresponding function values $f(x_i)$, the Lagrange interpolation formula for the value of f at the point x is

$$f(x) = \sum_{i=0}^{m} c_i(x)\ f(x_i)$$

where

$$c_i(x) = \frac{(x-x_0)\ (x-x_1) \ldots\ (x-x_{i-1})\ (x-x_{i+1}) \ldots\ (x-x_m)}{(x_i-x_0)\ (x_i-x_1) \ldots\ (x_i-x_{i-1})\ (x_i-x_{i+1}) \ldots\ (x_i-x_m)}$$

If the extrapolation option is specified, and the evaluation point is less than the initial tabular point or is greater than the final tabular point, then the first m+1 tabular values or last m+1 tabular values are specified for the extrapolation, respectively.

The choice of the order of interpolation should be based on knowledge of the actual nature of the function being interpolated. As is illustrated by the following example, the interpolated value is dependent on the order of the function and a higher order representation is not necessarily more representative than a low order interpolation.

Example: Consider the following table which gives values of THECOM as a function of TIME

TIME	THECOM
0.	0.
10.	0.
20.	1.
30.	1.
32.	2.
34.	2.5
36.	2.5
38.	2.
40.	1.
50.	1.

In the table below are the interpolated values of THECOM at the values of TIME listed for the indicated orders of interpolation.

ORDER \ TIME	5.	15.	25.	35.	45.	55.
1	0.	0.5	1.	2.5	1.	1.
2	-0.125	0.625	0.0417	2.562	0.0417	4.125
3	-0.250	0.5	0.330	2.562	-1.344	13.612
4	-0.969	0.931	-0.7[illegible]	2.562	-2.076	24.877
5	-4.506	2.267	-0.386	2.562	-2.524	38.020
6	-14.127	4.649	-1.160	2.562	-2.815	53.134
7	-34.603	8.082	-1.745	2.562	-2.815	53.134
8	-72.776	12.543	-2.174	2.562	-2.575	24.122

4.1.2 Multivariate Interpolation

The procedure for multivariate interpolation is analogous. Assume that f is a function of n independent variables, $x_1, x_2 \ldots x_n$, and assume that functional values of f are specified at the N_i+1 distinct points $x_i^0, x_i^1 \ldots x_i^{N_i}$ for the i-th variable. The multivariate interpolation process involves $(N_1+1)(N_2+1)\ldots(N_n+1)$ distinct points and the corresponding function values

$$f(x_1^{i_1}, x_2^{i_2} \ldots x_n^{i_n}), \quad 0 \le i_1 \le N_1, \quad 0 \le i_2 \le N_2 \quad \ldots$$

$$0 \le i_n \le N_n$$

The current GTS implementation requires that the total number of independent variables be less than or equal to 6. Also, the current GTS implementation does not permit the specification of an order of interpolation or the extrapolation or extend option with respect to each individual independent variable. Rather, the specified order of interpolation and any table extension option apply to all independent variables.

For the multivariate case, the interpolated values of the dependent variable are obtained by the multivariate Lagrange formula. Again, assume the order of interpolation is m. Then, at least m+1 points must be specified for each independent variable (i.e., $N_i \geq m+1$, $i = 1, 2, \ldots, n$). For each independent variable the m+1 points used in the interpolation formula are selected by the same procedure described for the single independent variable case.

Assume that an interpolated value of f is desired at an arbitrary point $x = (x_1, x_2, \ldots, x_n)$. With a slight abuse of notation, assume that $x_i^0, x_i^1 \ldots x_i^{m+1}$ represent the m+1 values of the i-th independent variable selection for the interpolation process. Thus,

$$x_i^0 < x_i^1 < \ldots < x_i^{k_i} \leq x_i \leq x_i^{k_i+1} < \ldots < x_i^{\ell}, \quad i = 1, 2, \ldots, n$$

The value of f(x) is computed by

$$f(x_1, x_2, \ldots, x_n)$$

$$= \sum_{i_1=0}^{\ell} \sum_{i_2=0}^{\ell} \cdots \sum_{i_n=0}^{\ell} C_{i_1}^1(x_1)\, C_{i_2}^2(x_2) \ldots C_{i_n}^n(x_n)\, f(x_1^{i_1}, x_2^{i_2}, \ldots, x_n^{i_n}),$$

where

$$C_{i_j}^j(x_j) = \frac{(x_j - x_j^0)(x_j - x_j^1)\ldots(x_j - x_j^{i_j-1})(x_j - x_j^{i_j+1})\ldots(x_j - x_j^{\ell})}{(x_j^{i_j} - x_j^0)(x_j^{i_j} - x_j^1)\ldots(x_j^{i_j} - x_j^{i_j-1})(x_j^{i_j} - x_j^{i_j+1})\ldots(x_j^{i_j} - x_j^{\ell})}$$

for

$$i_j = 0, 1, \ldots, \ell \text{ and } j = 1, 2, \ldots, n$$

As in single variable interpolation, the choice of the order of interpolation should be based on knowledge of the actual nature of the function being interpolated. Again, a higher order representation is not necessarily more representative than a low order interpolation.

4.2 Interpolation/Integration Interaction

If the interpolation operator is executed in conjunction with an integration operator (e.g., a trajectory simulation), then special processing related to the interaction of these two operators may be required. The exact relationship between the interpolation process and integration operator depends on such factors as the integration method, the integration step size, the order of interpolation, and the number of points in the table. Since both the integration operator and interpolation process make functional approximations to the tabular values, it is certainly desirable to have these approximations be similar. If, for example, the integration step size is large in comparison to the spacing of points in the table, then the function as represented by the interpolation table may differ significantly from the functional representation made as a part of the integration process. This situation may be especially troublesome with the Adams-Moulton integration method, which makes approximations based on information over several integration intervals. Within GTS, process control options related to tabular input attempt to insure this compatiblity. The table timing option attempts to alleviate this situation by requiring that each tabular value of the independent variable be an integration point. This option thus requires that each integration interval may not include a table interpolation point except as an end point.

The timing option does have the disadvantage of requiring that every tabular value is an integration point. The same accuracy, however, may be maintained by specifying the integration process include a few selected points; thus, the efficiency of the integration process is increased. Also, the Adams-Moulton scheme still requires information over several integration intervals even with the table timing option. That is, a Runge-Kutta restart is not necessarily performed at the tabular points. This procedure may not be desirable for certain types of tabular data such as those with sharp peaks. A GTL option permits the user to select individual points in the table, and the integration scheme will integrate to these points. A Runge-Kutta restart is then performed at these points. Note that for the Runge-Kutta method, the

table timing option is equivalent to specifying every tabular point for a restart. The current GTS implementation restricts the table timing and restart options tables for which the independent variable is time (i.e., trajectory time or time from an event).

Step tables require special logic distinct from the processing described for interpolation tables. Step tables provide a discontinuous representation of the tabular functions. Hence, if a function represented by a step table is part of the evaluation of the derivatives, then this function may introduce discontinuities into the derivatives of the differential equations being integrated. Consequently, the processing at tabular points of a step table is similar to the processing at phase points. Namely, the integration method must integrate up to such a tabular point and a Runge-Kutta restart must be made on the right side of such a point. Furthermore, since the function represented by the step table is discontinuous at such a point, it is necessary to evaluate the derivatives of the differential equations on the right side of such a tabular point.

REFERENCES

1. H. E. Pickett, A Contribution to the Thaumaturgy of Non-Linear Programming, Report No. ATR-71(59990)-1 (El Segundo, Calif.: The Aerospace Corp., August 1970).

2. J. T. Betts, An Effective Method for Solving Constrained Parameter Optimization Problems, Report No. TR-0073(3450-10)-1 (El Segundo, Calif.: The Aerospace Corp., 8 December 1972).

3. _____, Solving the Nonlinear Least Square Problem: Application of a General Method, Report No. TR-0074(4901-03)-3 (El Segundo, Calif.: The Aerospace Corp., 15 April 1974).

4. _____, An Accelerated Multiplier Method for Nonlinear Programming, Report No. TR-0075(5901-03)-5 (El Segundo, Calif.: The Aerospace Corp., 30 November 1974).

5. P. K. Henrici, Discrete Variable Methods in Ordinary Differential Equations (New York: Wiley and Sons, 1962).

6. C. W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations (Englewood Cliffs, New Jersey: Prentice Hall, 1971).

7. S. D. Conte and C. deBoor, Elementary Numerical Analysis: An Algorithmic Approach (New York: McGraw, 1972).

8. E. Isaacson and H. Keller, Analysis of Numerical Methods (New York: Wiley and Sons, 1966).